

AD-A116 808

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA

F/S 9/2

VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEHIC--ETC(U)

MAY 82 J LEE

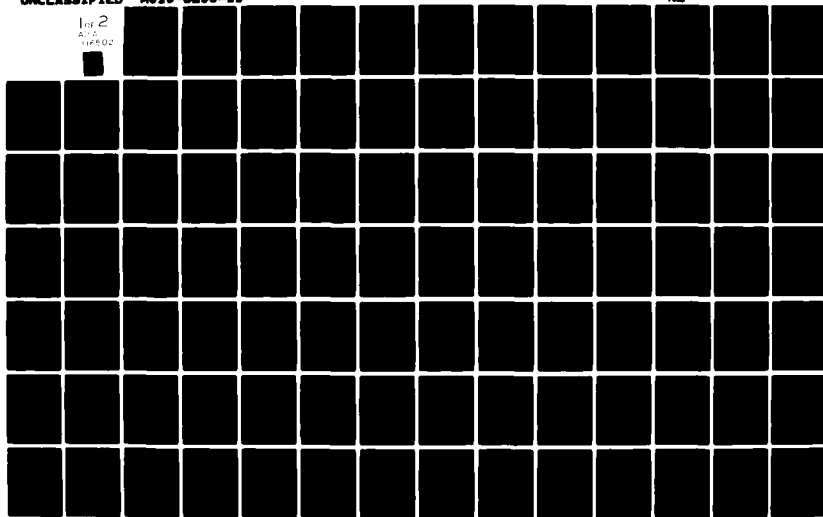
N00039-81-C-0063

W010-8208-10

ML

UNCLASSIFIED

for 2  
AD-A  
116 808

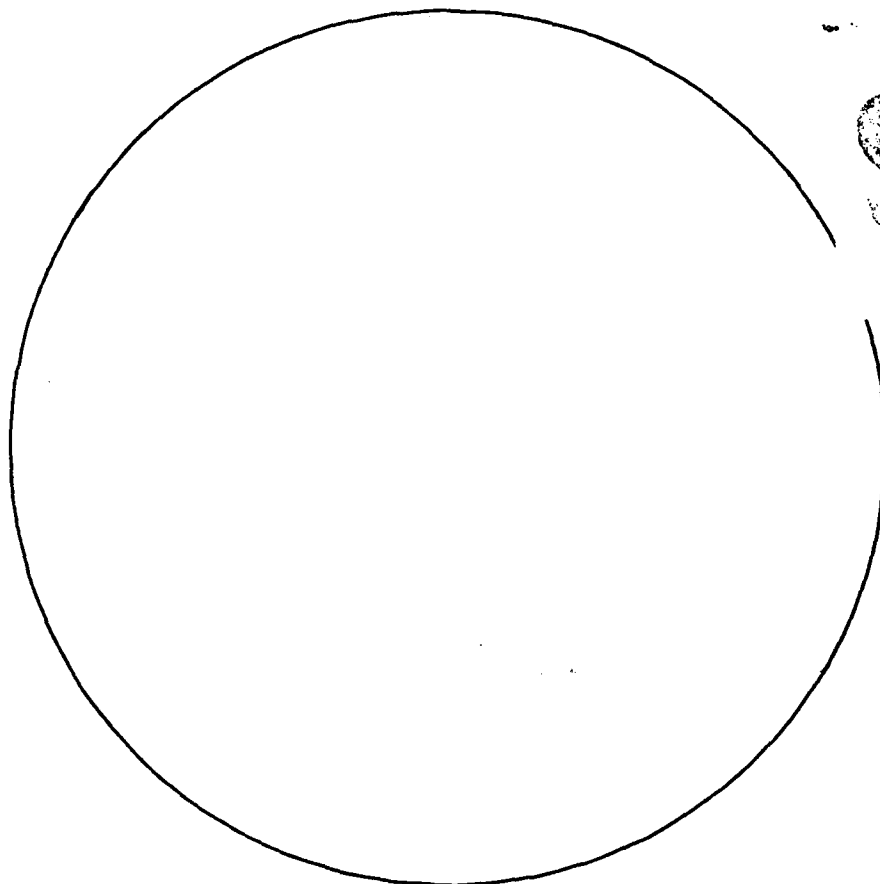


AD A116502

DTIC FILE COPY



(12)



DTIC  
ELECTRIC  
JUL 7 1982  
H

DISTRIBUTION STATEMENT A  
Approved for public release;  
Distribution Unlimited

**Center for Information Systems Research**

Massachusetts Institute of Technology  
Sloan School of Management  
77 Massachusetts Avenue  
Cambridge, Massachusetts, 02139

Contract Number N00039-81-0663 (MIT # 91445)  
Internal Report Number M010-8205-10  
Deliverable Number 6

12

VIRTUAL INFORMATION FACILITY  
OF THE INFOPLEX SOFTWARE TEST VEHICLE  
(PART I)

Technical Report #10

By

Jameson Lee

May, 1982

DTIC  
SELECTED  
JUL 7 1982

Principal Investigator:  
Professor Stuart E. Madnick

Prepared for:  
Naval Electronics Systems Command  
Washington, D.C.

DISTRIBUTION STATEMENT A  
Approved for public release;  
Distribution Unlimited

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #10	2. GOVT ACCESSION NO. ADP-116-722	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Virtual Information Facility of the INFOPLEX Software Test Vehicle		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER M010-8205-10
7. AUTHOR(s) Jameson Lee		8. CONTRACT OR GRANT NUMBER(s) N0039-81-C-0663
9. PERFORMING ORGANIZATION NAME AND ADDRESS Sloan School of Management, MIT 50 Memorial Drive, Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE May 1982
		13. NUMBER OF PAGES 180
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)  database computer, database management system, Software Test Vehicle, hierarchical system, virtual information		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)  This report describes the software design and implementation of the front- end for the Virtual information facility of the INFOPLEX database computer. It is part of a major effort to develop a software simulation, called Software Test Vehicle, for the underlying architecture of INFOPLEX. The virtual information facility is a single level of operations situated within the Functional Hierarchy. It supports the use of virtual information, a virtual entity based on procedural relationships and derivations from		

physically recorded data. Upon completion, this facility will be integrated within the current implementation of the STV for the INFOPLEX Functional Hierarchy which lacks the support for virtual information processing.

Accession	
NTIS	<input checked="" type="checkbox"/>
DTIC	<input checked="" type="checkbox"/>
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A	



Virtual Information Facility  
of the INFOPLEX Software Test Vehicle

by

JAMESON LEE

Submitted to the Department of Electrical  
Engineering and Computer Science in May,  
1982, in partial fulfillment of the  
requirements for the degree of  
Bachelor of Science

Abstract

This thesis is a software design and implementation of the front-end for the Virtual Information Facility of the INFOPLEX data base computer. It is part of a major effort to develop a software simulation, so called a Software Test Vehicle, STV , for the underlying architecture of INFOPLEX.

INFOPLEX is a hierarchical architecture for data base computers, based on functional decomposition of data base operations. It is a current research project of the Information Systems Group at M.I.T.'s Sloan School of Management. Within the INFOPLEX architecture, a functional hierarchy of information management functions is built on top of a storage hierarchy of information storage functions. These two independent hierarchies are further divided into many sub-levels, each of which is devoted to a more specific function of data base activities.

The virtual information facility is a single level of operations situated within the functional hierarchy. It supports the use of virtual information, a virtual entity based on procedural relationships and derivations from physically recorded data. Upon completion, this facility will be integrated within the current implementation of the the STV for the INFOPLEX functional hierarchy which lacks the support for virtual information processing.

Thesis Supervisor: Professor Stuart E. Madnick  
Sloan School of Management, M.I.T.

## Contents

	Page
Title.....	1
Abstract.....	2
Acknowledgement.....	4
Contents.....	5
List of Figures.....	8
Chapter 1 Introduction.....	9
1.1.0 INFOPLEX Overview.....	9
1.1.1 Concept.....	10
1.1.2 Infoplex Architecture.....	10
1.1.3 Functional Hierarchy.....	12
1.1.4 Research Issues.....	12
1.2.0 Thesis Objectives.....	12
1.2.1 Background.....	14
Chapter 2 Virtual Information.....	16
2.1.0 Concept.....	16
2.2.0 Classification.....	16
2.2.1 Factored Facts.....	16
2.2.2 Computed Facts.....	17
2.2.3 Inferred Facts.....	18
2.3.0 Specification.....	19
2.4.0 Merits.....	19
2.5.0 Approach.....	21
Chapter 3 Functionalities.....	23
3.1.0 Underlying Data Model.....	23
3.2.0 Active Workspace.....	24
3.3.0 Permanently Defined Virtual Information.....	25
3.4.0 Adhoc Virtual Information.....	25
3.5.0 Notion of a Transaction.....	26



3.6.0	Virtual Attributes.....	26
3.7.0	Conditions on Real or Virtual Attributes.....	28
3.8.0	Virtual Entity Sets.....	28
3.9.0	Generalized Macro Facility.....	29
3.10.0	Extended Functionalities.....	30
3.10.1	User Dependent Virtual Definitions.....	30
3.10.2	Inferred Facts of Undesignated Indirection....	31
Chapter 4	Program Structure.....	32
4.1.0	Module Description.....	32
4.1.1	User-Interface.....	34
4.1.2	Buffer.....	36
4.1.3	Activity Coordinator.....	41
4.1.4	Tokenizer-Processor.....	42
4.1.5	Language Design and Specification.....	45
4.1.6	Finite-State-Automaton (Machine).....	45
4.2.0	Internal Global Variables.....	46
Chapter 5	Language Illustration and Specification.....	48
5.1.0	Data Base Statements.....	48
5.1.1	Define Statements.....	48
5.1.2	Adhoc Statements.....	49
5.1.3	Listdef Statements.....	50
5.1.4	Retrieve Statements.....	50
5.2.0	Buffer Commands.....	58
5.2.1	Command Syntax.....	59
5.3.0	Formal Description of Data Base Language Grammar.	61
5.3.1	BNF Supplement.....	64
Chapter 6	Finite-State-Machine.....	66
6.1.0	Configuration.....	68
6.2.0	Match-Action-Next_State Rules.....	69
6.3.0	Action Routines.....	76
6.4.0	Listing.....	79
Chapter 7	Major Design Issues.....	90
7.1.0	Form of Storage for Virtual Definitions.....	90

7.2.0	Parser Structure.....	92
7.3.0	Program Control Structure.....	92
7.4.0	Interactive Editor.....	93
7.5.0	Language Design.....	94
Chapter 8	Conclusion.....	96
	Bibliography.....	98
	Appendix: .....	99
Programs Listing		
	USER-INTERFACE (USINT).....	100
	BUFFER (NEWBUFF).....	106
	ACTIVITY COORDINATOR (ACTCRD).....	128
	TOKENIZER-PROCESSOR (TKNIZE).....	136
	DATA STRUCTURES .....	148
	FINITE-STATE-MACHINE Rules.....	151
	A Very Simple Sample Session.....	161

## List of Figures

	Page
1.1 INFOPLEX Architecture	11
3.1 Entity Data Model	24
3.2 Sample Data Graph	27
4.1 Module Flow Chart	33

## 1.0.0 INTRODUCTION

INFOPLEX DATA BASE COMPUTER is a current research project of the Information Systems Group at M.I.T.'s Sloan School of Management. It proposes a new architecture whose objectives are to provide substantial improvements in information management performance over conventional computer architectures, and to provide highly reliable support for very large and complex data bases.

### 1.1.0 INFOPLEX OVERVIEW

Progress of modern society has put increasingly more new and challenging demands upon the capability and performance of information storage, retrieval, and management. Conventional computers, whose architecture is designed primarily for computational objectives, are not suited to meet the requirements of these new demands. Efforts have been made in four different areas to build computer systems which will suit our information needs today, and in the future: (1) new instructions through microprogramming, (2) intelligent controllers, (3) dedicated computers for data base operations, and (4) data base computers. INFOPLEX is a research project belonging to the fourth category.

#### 1.1.1 CONCEPT

INFOPLEX employs the concept of hierarchical decomposition which organizes information management functions into a functional hierarchy, and the physical memory management functions into a storage hierarchy (Madnick 78); both hierarchies consist of many independent levels of operation, each of which supports a different set of information or storage management functions through the use of multiple microprocessors.

#### 1.1.2 INFOPLEX ARCHITECTURE

As stated previously, INFOPLEX is an architecture for data base computers based on hierarchical decomposition. A functional hierarchy of information management functions is built on top of a hierarchy of information storage functions. Both hierarchies are further divided into many functionally independent levels of operation, each of which is to be supported by a set of micro-processors operating in parallel with one another. A global Communication Bus coordinates inter-level transmission of data. This hierarchical architecture exploits the advantages of functional modularity of operations, and of parallel processing of micro-processors to systemize data base activities and to achieve a prescribed level of efficiency. A graphical illustration of this architecture is presented in figure 1.1 .

# INFOPLEX Architecture

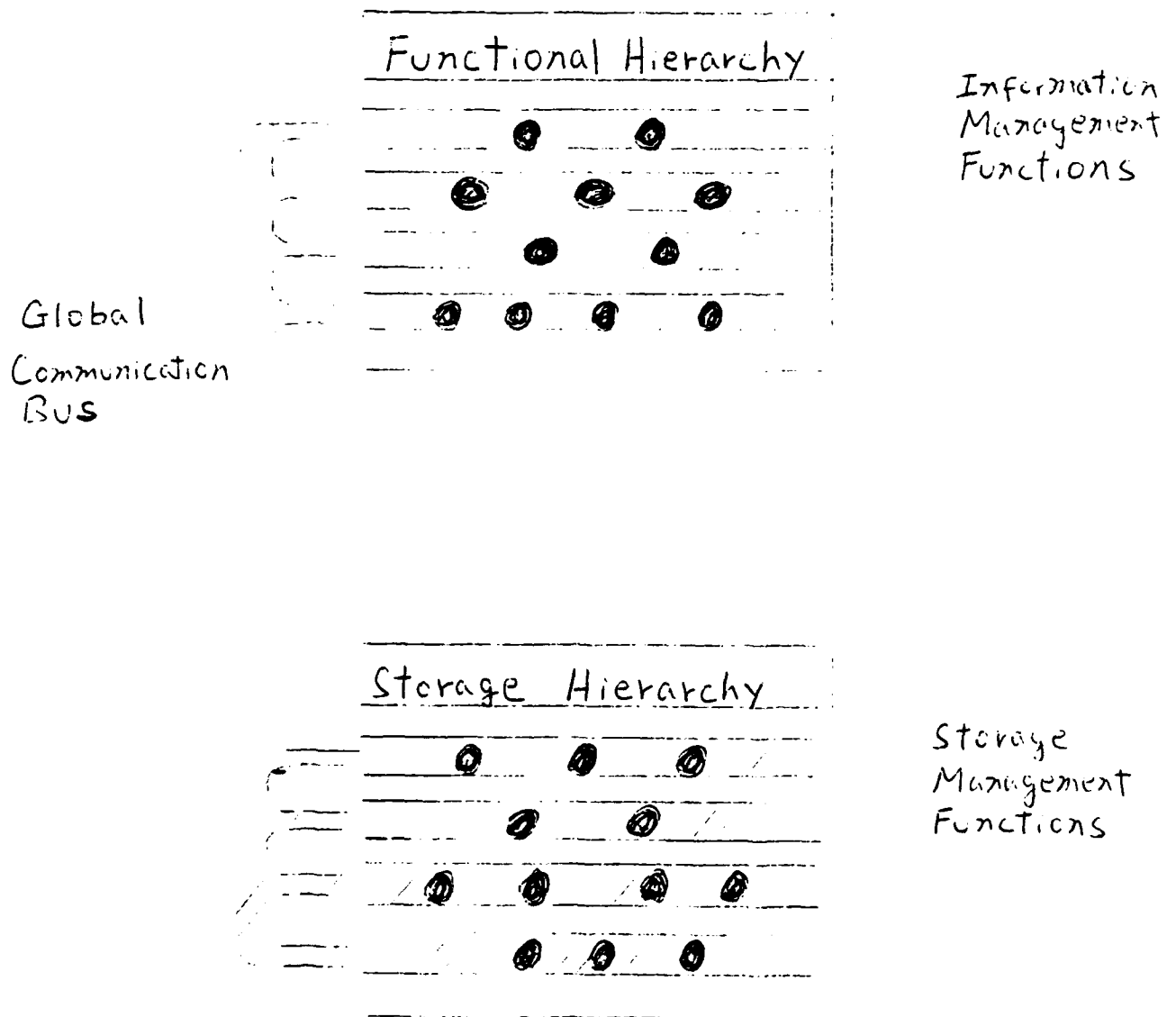


Figure 1.1

### 1.1.3 FUNCTIONAL HIERARCHY

Current architecture of the functional hierarchy (Hsu 1982) with respect to data abstraction consists of four separate levels: (1) external level, (2) conceptual level, (3) entity level, and (4) internal level. A part of the conceptual level is a virtual information facility (Hsu 1982). These four levels of information management are highly independent of one another, and each is responsible for a different but necessary phase of information processing in a data base computer.

### 1.1.4 RESEARCH ISSUES

Major efforts of INFOPLEX research are devoted to the design, modeling, and evaluation of an optimal decomposition strategy for both the functional and memory hierarchy of information management and storage operation, and also to the study of an associated distributed control mechanism. This control mechanism would be used to coordinate the activities of and inter-level communications within the hierarchies.

### 1.2.0 THESIS OBJECTIVE

This thesis shares a joint mission with a concurrent thesis by Peter Lu. The two theses are entirely separate in

functionalities, but closely related and dependent upon one another for a complete software simulation of the virtual information facility on the INFOPLEX data base computer architecture. This facility would incorporate the design and implementation of two sub-levels of the INFOPLEX functional hierarchy, the virtual information level, and an user interface level which is tailored for the use of virtual information processing.

This thesis is responsible for fulfillment of the front-end objectives of the joint mission; the front-end objectives include the design and implementation of the following:

- a) A data base language to support virtual information
- b) A finite state machine to parse data base statements written in this language
- c) A user-interface tailored to the use of virtual information.
- d) A processor to process the creation, listing, and modifications of virtual definitions, as well as the substitution of these definitions into data base statements in actual use.



This processor would also be responsible for transforming data base statements into a chain of tokens, each of which would include an indicator describing the classification of the token according to a prescribed classification scheme.

The combined objectives of this "front-end" and Peter Lu's "back-end" would fullfill our joint mission as mentioned earlier, namely, to construct in software a virtual information facility with its own user interface, from here on referred to as VIFI, Virtual Information Interpreter.

#### 1.2.1 BACKGROUND

In the three short months in which VIFI was developed, we labored and wished to exhibit a certain degree of professionalism in its design and implementation. The merits of modular programming, of innovative algorithms, of performance efficiency, of functional capabilities, of user-friendliness of the proposed data based language, of program organization and flexibility, and even of consistencies in programming style were evaluated against time and labor limitations. A serious attempt was made to incorporate all of these characteristics into our Virtual Information Interpreter.

While making these considerations, many sleepless nights of unceasing arguments plagued the two developers; it was the intrinsic dissention between the idealist and the pragmatist. At a certain point, such disagreements grew to be so severe that it appeared to have left an unpleasant mark on a very close and strongly bonded friendship. However, a lesson of humanity was learned from this experience, and our cherished friendship would continue to grow, and become stronger than never before, because we have acknowledged a feeling of faith and destiny which was manifested through this experience. I am expressing this sentiment here because I consider it the most personally meaningful and lasting reward of this thesis.

## 2.0.0 VIRTUAL INFORMATION

### 2.1.0 Concept

The concept of virtual information in data base systems has been developed and examined in earlier research of the Information Systems Group. Basically, there is a spectrum of the kinds of information which may be retrieved from a data base. Along this spectrum, pure data occupy an extreme on one end, and pure algorithms occupy the extreme on the other. In between these two extremes are the information which may be derived from a combination of data and algorithms; such information are dynamic and procedural in nature, and are referred to as Virtual Information.

### 2.2.0 CLASSIFICATION

Virtual information may be categorized into three major classes: factored facts, inferred facts, and computed facts. Together, these three classes of virtual information and combinations there of, constitute the portion of the information spectrum between the two extremes of pure data and pure algorithms.

#### 2.2.1 FACTORED FACTS

Factored facts, subsets of data elements, based on certain prescribed conditions, or so called predicates, of attribute values, are often very valuable in structuring information in a useful manner. For instance, if a certain data base maintains records of weight, hair color, and salary for a group of employees, it may be useful to select from this group those individuals who share a certain condition on their attribute values, such as having black hair, making a salary greater than 8 dollars per hour, or weighing over 300 pounds. It is important that users of information should be able to access information independent of the particular factoring involved; this would imply the ability to support multi-level factoring, or repeated factoring of data.

#### 2.2.2 COMPUTED FACTS

Computed facts are those information which are obtainable through the application of particular computational algorithms and operators on data or groups of data. These operators include arithmetic, comparative, boolean, and other kinds of functions. In the very least, computed facts include those pure data manifested in a different form, with a different unit of measure, or an alias name. For instance: a user may define a virtual age attribute to be the difference between the current year and a person's birth-year, a virtual rectangular area

attribute to be the length multiplied by the width, or an attribute value in the unit of inches to be 12 times the attribute value in the unit of feet. In this sense, transformations between different units of measure are intrinsic to the operations of computed facts.

### 2.2.3 INFERRED FACTS

Inferred facts pertain to implicit relationships which the data base system may arrive at through certain levels of indirection. In other words, a path, although indirect, does exist which leads to the desired data in storage. There are two ways by which the system on its own can support this kind of virtual information. The first method is by an exhaustive search of all possible paths, and the second is the application of a certain degree of artificial intelligence to deduce a viable path to the target data. Well, the first method is unbounded in computing time, and even when a path is found, it may not be the correct path; the second method is far fetched at this time. Therefore, we will give our attention to a different but comparable set of inferred facts which is implementable, and we give it the name Pseudo Inferred Facts. Pseudo Inferred Facts are exactly the same as inferred facts except that all the indirections will be explicitly designated by the user. With this strategy, exhaustive search is not necessary, artificial intelligence is not necessary, and the specified path would

always be the designated and correct path. For instance, the Uncle relationship may be defined as the application of the Brother relationship after the application of the Mother relationship.

#### 2.3.0 SPECIFICATION

Users of information, through the virtual information facility, define their own working environment and the manner in which they would like to use the physical and underlying data. Such definitions of virtual information may be accomplished through a virtual information definition language. The virtual information facility would accept virtual information definitions and their modifications in the definition language, and respond to virtual information retrieval requests through a separate virtual information retrieval language.

#### 2.4.0 MERITS

There are several major merits in the support of virtual information in a data base system. It is dynamic in nature because its definition may be created, deleted, and modified readily; its definition applies to all instances of data where it may apply, and yet there is but only one copy of this definition stored in the system. By facilitating the ease of

modification, it enhances data base flexibility, by eliminating redundant physical records, it contributes to more consistent data, and by being procedural in nature, it enhances information accuracy through the delay in the evaluation of data which vary over time or other changing factors until their time of use. These kinds of merits are based on virtual information's association with procedural relationships. For instance: the stored algorithm for computing age would eliminate the need to update the age attribute day by day if it were physically stored, and would be applied to calculate anyone's age, thus eliminating redundancy of stored information.

Virtual information also conserves the use of vast amounts of physical storage. It makes unnecessary the storage and maintenance of those information which may be derived upon request. This raises the issue of Time/Space trade-off, which should be seriously considered when deciding which kinds of fundamental data are or are not to be physically stored. Derivation upon requests will have the added cost of derivation; therefore, those information which will be used many times and are also difficult to derive may be the best kind of data to be physically stored; those information which is seldomly used and easy to derive may be the best kind of data not to be physically stored. Furthermore, the situation is made even more complex as we realize that the definitions themselves will require the use of physical storage. Thus, it wouldn't be an

easy task to decide which kinds of data are to be derived, or to be actually stored.

The definition of virtual information on a per user basis would simulate an entire virtual data base for each individual user. Each one would be free to tailored the data base to his own preferred view or use through the virtual information definitions. A particular set of virtual definitions may be very useful for one group of users, and another set for another group of users. In this sense, each one has gotten a data base suited for his own use while not affecting anybody else's usage of the data base. A logical extension of this scenario is to implement access control mechanisms such that users may establish a controlled sharing of sets of virtual information definitions with one another; the data base administrator may monitor all such sharing to prevent unauthorized access to a certain set of virtual information functions. However, in a scenario as such, a separate catalogue would have to be maintained for each and every user, and considerable catalogue management would be required. Such is the cost for this individually user-tailored data base functionality, a secondary merit of the use of Virtual Information.

#### 2.5.0 APPROACH



The concept of virtual information leads directly to a functional approach to data bases. A virtual information facility would be treated as a collection of functions, and retrieved data would be regarded as functional values. Virtual information requests correspond to function invocations; this functional approach to information readily supports procedural relationships on which based the concept of virtual information. As a result, a virtual information facility is likely to resemble very much a language interpreter which accepts functional definitions and respond to functional invocations with specified arguments.

### 3.0.0 FUNCTIONALITIES

There are numerous functionalities to a virtual information facility, each of which may be implemented to a varying degree of completeness. Although it may be desirable to implement all the functionalities there are wherever possible, it may be too impractical and less than meaningful for the initial version of the implementation. Thus, we have not implemented the One Data Base per user feature of virtual information capabilities which we have described in the previous chapter. Later portions of this chapter would describe the functionalities of virtual information which we did implement; surely, not all of these implementations would be without room for further refinement, even though they already include an extensive set of virtual information capabilities.

### 3.1.0 UNDERLYING DATA MODEL

The virtual information facility lies on top of the entity set level of the functional hierarchy. In this level, the data base is seen as a network of entity sets and their attributes. Each entity set may have a varying number of attributes, some of them being value attributes and others being entity attributes. (Hsu 1980) The value attributes include a set of attribute values, and the entity attributes represent

relationships leading to other entity sets. Figure 3.1 briefly illustrates this model.

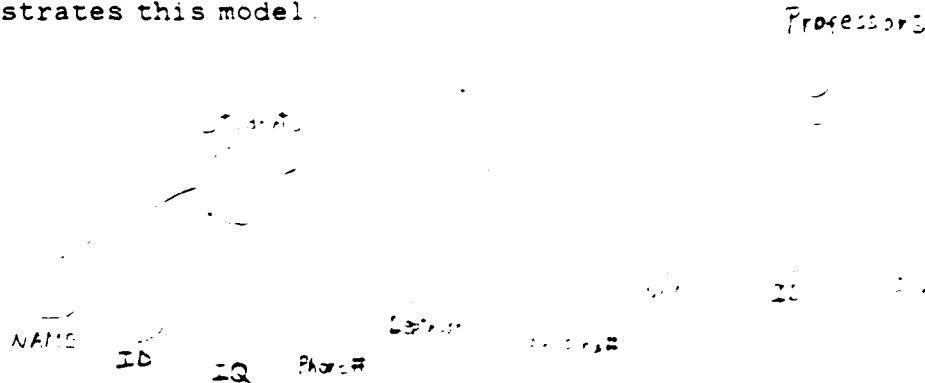


Fig 3.1

### 3.2.0 ACTIVE WORKSPACE

We have developed an active workspace which incorporates a line editor with full screen display, through which user commands may be issued. The workspace consists of two buffers, an execution buffer, and a transaction buffer. The transaction buffer withholds many data base statements which will be executed sequentially when the transaction buffer is executed. The execution buffer holds a single data base statement and will be automatically executed when a data base statement is completed. A number of buffer commands is created to manipulate buffer contents. The details of these commands as well as the data base statements will be illustrated in chapter 5.

### 3.3.0 PERMANENTLY DEFINED VIRTUAL INFORMATION

Permanent virtual information may be defined through the Define statement. Such definitions will be stored in a global dictionary, or so called catalogue, in the form of character string, and will remain there until explicitly removed or over-written by a different definition. Examples may be found within chapter 5.

### 3.4.0 ADHOC VIRTUAL INFORMATION

Virtual information definitions may be derived for only the duration of a single transaction. When all statements within the transaction are executed, the adhoc dictionary would be erased. Within the transaction, adhoc definition may be created, deleted, as well as modified at any time. With this feature, each transaction would be associated with a catalogue of its own, and would not interfere with the concurrent activities of other transactions executing in parallel. At this stage, we do not support concurrent transactions, but adhoc definition capability is still useful in the principle of transactions. Surely, the permanent dictionary would also be accessible from within each transaction.

### 3.5.0 NOTION OF A TRANSACTION

A transaction is a body of executable statements joined together within a single context. This context is provided by the adhoc dictionary associated to the particular transaction. A transaction is created within the transaction buffer, and will remain there until it is explicitly over-written, erased, or executed. Merits of this transaction concept are threefold: a) a group of statements which collectively does a certain task may be consolidated to exhibit logical unity. b) a shared context may be created and maintained for each transaction, a sign of transactional modularity and independence from one another. c) the execution of the consolidated operations in a transaction may be put off until a more opportune moment, by which time new permanent or adhoc virtual information definitions may be defined either to supplement or to replace existing definitions.

### 3.6.0 VIRTUAL ATTRIBUTES

Virtual attributes equated to the results of computational algorithms acting on available data or of designated indirect references may be explicitly defined through the Define data base statement. This feature incorporates the support for Computed Facts as well as for Pseudo Inferred Facts. For instance, the following is the definition and usage of two virtual attri-

butes, income and ship-country, a computed fact, and a pseudo inferred fact.

Define income as salary - expenses ;

Retrieve ({teachers}) by ({V0} name, income) ;

The foregoing retrieve statement returns two vertical columns of data. The first column being teacher's name, and the second column being their corresponding incomes.

Define ship-country as ~~ship~~ company ( country ( name )) ;

Retrieve ({ship}) by ({v0} name, ship-country) ;

This foregoing retrieve statement returns two columns of data, the first being individual ship names, and the second being the name of the country to which the ship belongs to. The entity diagram for this scenario is as follows:

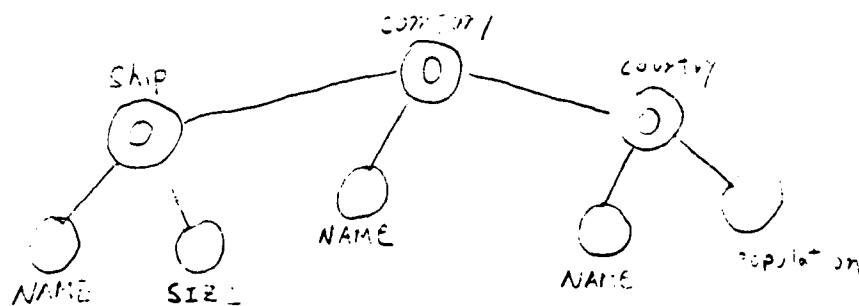


Fig 3.2

### 3.7.0 CONDITIONS ON REAL OR VIRTUAL ATTRIBUTES

Arbitrary conditions on real or virtually defined attributes may be defined by INFOPLEX users as the shared 'condition' on their data values from which factored facts may be later constructed. For example:

```
Define old as age > 70 ;
```

```
Define rich as assets > 1000000 ;
```

```
Retrieve ({people}where(rich and old)) by ({VO} name);
```

The foregoing retrieve statement would return a list of names of those people whose age > 70 and assets > 1000000.

### 3.8.0 VIRTUAL ENTITY SETS

Aside from virtual attributes, we also support a basic notion of virtual entity sets. We recognize two kinds of virtual entity sets:

a) Union or intersection of real or previously defined virtual entity sets based on their real and virtual attribute values.

b) Subsetting of real or virtual entity sets based on certain conditions on their real and virtual attribute values.

For instance:

Define ClassAB as {ClassA} MU (Name) {ClassB} ;

ClassAB is defined as the result of a multiple-union operation on entity sets ClassA and ClassB, based on a common attribute called Name.

Define RichMen as {Men} where (assets > 1000000) ;

RichMen is defined as a virtual subset of the set Men, based on the values of its asset attributes.

The complete set of union and intersection operators as well as the cartesian product operator between entity sets is illustrated within chapter 5. Also, refer to chapter 5 for details of the capability to specify various conditional predicates on attribute values.

### 3.9.0 GENERALIZED MACRO FACILITY

Users will be able to define arbitrary definitions and to give them specific names by which the definitions may be referred to and later substituted into data base statements. In this sense, the define statement may be used not only to



define virtual attributes, virtual entity sets, but also random definitions as well even if the definitions are seemingly incoherent without the proper context. When a retrieval statement is to be executed, all words within the statement are first checked against a list of stored definition names; any matching definition would be recalled from the dictionary and put in the place of the matching definition name in the retrieval statement. Chapter 5 includes a detailed description of such usage.

#### 3.10.0 EXTENDABLE FUNCTIONALITIES

##### 3.10.1 USER DEPENDENT VIRTUAL DEFINITIONS

This particular functionality is not difficult to implement, but it may be unnecessary at this stage of the project. It simply would require a separate catalogue for each user which includes an access control list, proper search rules including default situations, and adequate coordination and control mechanisms to manage the various catalogues. It would increase the cost in terms of time and space efficiency. Thus, we have not included this functionality in this version of virtual information implementation. Nevertheless, if circumstances in later time are such that the support for user dependent catalogues is so desirable as to more than compensate for its cost of implementation, this functionality may be added readily.

### 3.10.2 INFERRED FACTS OF UNDESIGNATED INDIRECTION

Inferred facts with undesignated indirection, rather than pseudo inferred facts with designated indirection, is likely to have tremendous costs in system performance whenever it is to be implemented. As previously stated, this would require either an exhaustive search or a certain level of artificial intelligence, both of which require large amounts of resources in computing power, storage and time. Furthermore, in order to verify that the indirection the system chooses at each step along the way is correct, the user has to monitor the computer decisions interactively; this defeats the original purpose of not having the user to designate his intended path of indirection. Thus, it seems very doubtful that this functionality will ever be implemented unless the requirements for user monitoring of the decision process is somehow eliminated.

#### 4.0.0 PROGRAM STRUCTURE

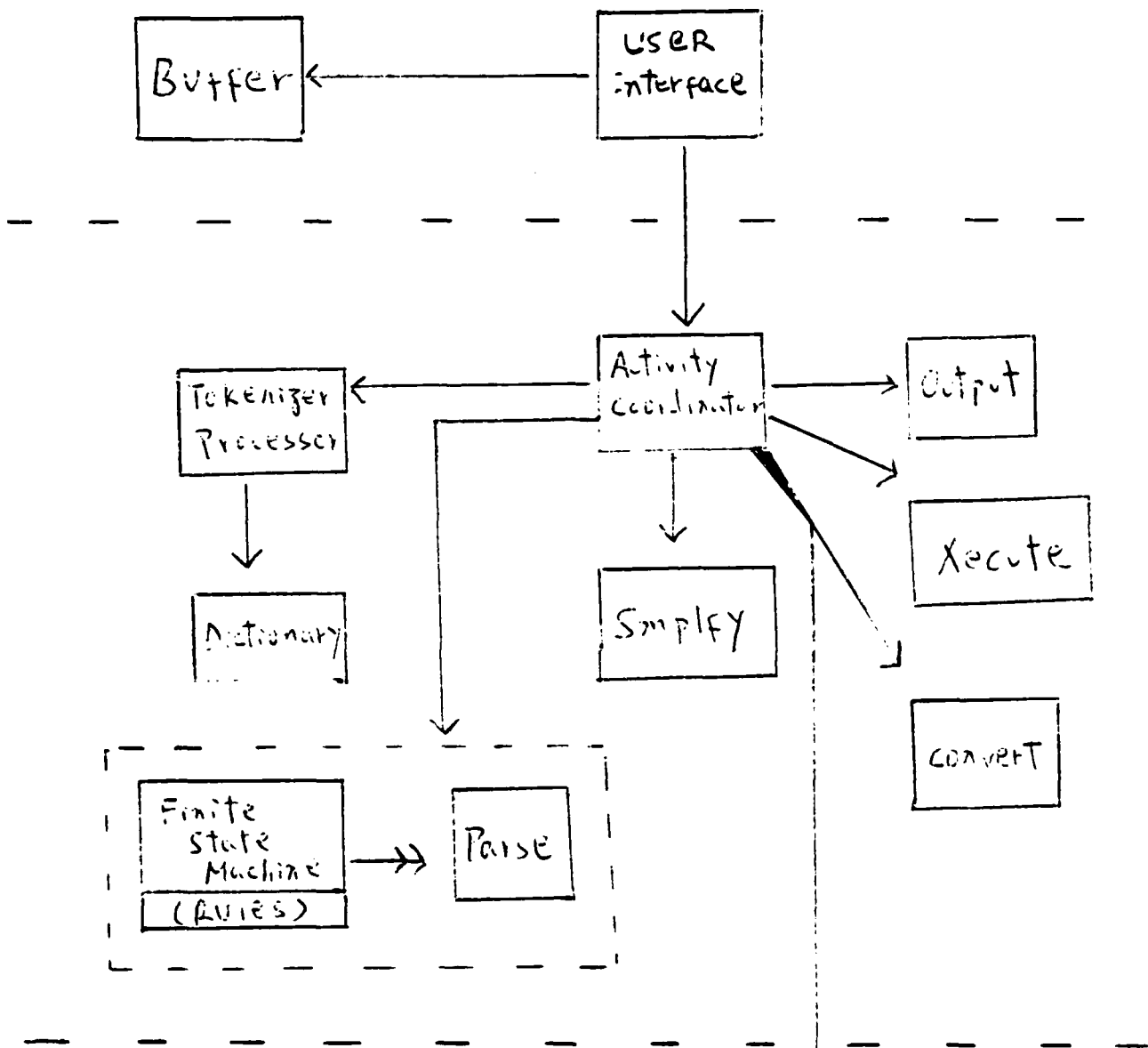
Our implementation is done with special attention to modularity. Each primary module incorporates numerous internal sub-modules whose very existence are not known nor relevant to the implementation of other primary modules. Aside from the PL/1 modules, we have designed a data base language, and a finite state push-down automaton, each of which will be categorized as a single module as well. Figure 4.1 illustrates the control structure and data flow of all the modules in our implementation. Single arrow heads in the diagram represent control structure transitions and double arrow heads represent data flow.

#### 4.1.0 MODULE DESCRIPTION

The front-end as designed and implemented in this thesis includes the following modules:

- (1) USER-INTERFACE
- (2) BUFFER
- (3) ACTIVITY-COORDINATOR
- (4) TOKENIZER-PROCESSOR
- (5) LANGUAGE DESIGN and SPECIFICATION
- (6) FINITE-STATE-PUSH-DOWN AUTOMATON (MACHINE)

# Module Diagram



All programs are written in PL/1 under CMS operating system on IBM VM/370.

#### 4.1.1 USER-INTERFACE

This module is named USINT as a PL/1 program. It is currently the options-main program of the entire virtual information facility. It diverts control to one of two INFOPLEX implementations, one of which includes our virtual information facility, and the other does not. Once the implementation with virtual information facility is selected by the user, this module serves as the communication link between the BUFFER module which interacts with the user and the ACTIVITY COORDINATOR module on the lower level which supervises the execution of data base statements.

This module has five internal routines:

- (1) XBUFF
- (2) GETS
- (3) REPLACE (internal to RETDSPLY)
- (4) RETDSPLY

The XBUFF routine strips individual executable data base statements one by one off the buffer, and pass them down to the next level for further processing.

The GETS routine is a generalized tool which actually does a substring command from the first character of a given string to the first occurrence of a given character. After the execution of this routine, the portion of the original string up to and including the given character would be eliminated.

For instance:

```
str1 = 'abc$def'
```

Gets (str1, '\$') would return 'abc' and the value of str1 becomes 'def' .

The REPLACE routine is also a generalized tool to replace all occurrences of a given varying character string of length two or less, by another varying character string of length two or less.

For instance:

```
str1 = 'abc,def'
```

Replace (str1, ',', '55') would change the value of str1 to 'abc55def'

The RETDSPLY routine simply displays the current retrieval statement which is being processed to indicate the correspondence of subsequent outputs to this particular statement. It

makes numerous calls to REPLACE because many characters have been previously translated to enable the application of the finite state machine.

#### 4.1.2 BUFFER

The BUFFER module is named NEWBUF as a PL/1 program. It continuously interacts with the user during a virtual information session. It has a transaction buffer which corresponds to the "transaction" concept of virtual information, and which would accumulate successive data base statements until the entire transaction is to be executed. It also has an execution buffer which would be automatically executed upon the completion of a single executable data base statement. The word "execution" in the context of this module simply means the return of control to the module which called it, USER-INTERFACE. When returning control to the caller, if the transaction buffer is to be executed, then the transaction buffer content will be moved to the execution buffer, and if user requested the termination of the virtual information session, then a control bit passed to it from USER-INTERFACE would be set.

In order to facilitate the using of virtual information, this module incorporates a full screen but simple line editor which is coupled with the existing buffer commands. Buffer commands enable the moving of data from buffer to buffer, execution of

either buffer content, input of buffer content from a CMS file, saving of buffer content to a CMS file, and termination of the active session. The editor commands are INSERT, DELETE, and TOPLINE; they enable a simple editing of the transaction buffer content. The usage of these editor commands and buffer commands is described in Chapter 5.

In essence, this module establishes the Active Workspace environment described earlier. It is the primary module of the external level of the functional hierarchy, developed specifically for the use of virtual information facility on the next lower level.

This module has the following internal routines:

- (1) LDSPCH
- (2) BUILDBUFF
- (3) TRNSLATE
- (4) FINPUT
- (5) KBLKS
- (6) GETS
- (7) NEXTWORD
- (8) RMVFBKLS (internal to NEXTWORD)
- (9) FSAVE
- (10) WAIT
- (11) REPLACE
- (12) HELP



- (13) SETMKS
- (14) BDISPLAY
- (15) DELTE
- (16) WTHNLM

The LDSPCH routine contributes to the format integrity of each user inputted line by constructing a header which is concatenated to the front of each line. The header begins with a "@" character, which is succeeded by a numeric character string representing the number of leading blank characters in this line, and ends with a ":" character. In this manner, all leading blank characters of each line may be removed. The advantages of using such a header are twofold; not only can storage be conserved, but also a fixed structure be imposed on all user inputs to reduce complexity.

The BUILDBUF routine constructs either the execution or the transaction buffer one line at a time from each user input line. Markers on either buffer is repositioned to enable proper editor display. It returns a boolean value of "true" if there is at least one completed data base statement in the current buffer.

The TRNSLATE routine checks for missing quote terminators for character string constants and back-slash terminators for comment lines. It also translates ";" characters within com-

ments to "%2" and consecutive single quote characters within character string constants, representing an actual quote character, to "%1". Such translations are necessary to avoid ambiguity and complications in the input recognition stages of the process.

The FINPUT routine serves to input transaction buffer content from a CMS file whose file name is "file" and file type is given by the user through the "finput" buffer command. Original transaction buffer content is erased. Characters "%", ",", ":", "@", and "&" are replaced by "%4", "%0", "%3", "%5" so that they would not interfere with finite state machine command language.

The KBLKS routine serves to remove all leading blank characters from the current input line, and also to keep the number of them removed in variable "ldspaces".

The GETS routine is a general tool as described earlier within the USER-INTERFACE module.

The NEXTWORD routine returns the sequence of characters in the input line up to but not including the first blank character. If a blank character is not found, the entire input line is returned. Comments are automatically removed and are not recognized as part of an input line.

The RMVFBLKS routine is internal to NEXTWORD; it serves to remove the blank characters preceding each word in the input line. Its name stands for remove-front-blanks.

The FSAVE routine is the counter part to the FINPUT routine. It writes the current transaction buffer content into a CMS file whose file name is "file" and file type is given by the user through the "fsave" buffer command. The characters translated by FINPUT and TRANSLATE routines are restored before they are written into the CMS file.

The WAIT routine serves as a time delay to hold messages to user on display terminals long enough to be readable by the human eye.

The REPLACE routine is a general tool as described earlier within the USER-INTERFACE module.

The HELP routine displays a brief explanation of each buffer command to the display terminal.

The SETMKS routine sets or resets markers in either the execution or the transaction buffer for buffer-display purposes of the full-screen line-editor. The name "setmks" stands for "set-markers".

The BDISPLAY routine serves to display the contents of both the execution and the transaction buffer. It implements the full-screen characteristic of our line-editor.

The DELTE routine implements the delete-line function of the editor. The transaction buffer markers are properly reset after each invocation of this routine.

The WTHNLM routine serves to verify the logical correctness of editor command correctness. If the parameters are out of current buffer boundaries, then the routine will return 'O'B .

#### 4.1.3 ACTIVITY COORDINATOR

This module coordinates all activities on the level of virtual information processing. It directs the moving of program control through various modules on this level. A number of debugging tools which prints out various trees, token chains, and tables are included within this module, and can be used in times of need by inserting a "call" statement any where within the module.

This module contains the following internal routines:

- (1) GETS
- (2) PRINTT

- (3) PRINTM
- (4) PRINTX
- (5) PRINTE
- (6) PRINTR

The GETS routine is a general tool as described earlier within the USER-INTERFACE module.

The PRINTT routine is a debugging tool which can be used to print the chain of input tokens.

The PRINTM routine is a debugging tool which can be used to print a snap shot of the finite state machine.

The PRINTX routine is a debugging tool which can be used to print the execution tree.

The PRINTE routine is a debugging tool which can be used to print the entity set table.

The PRINTR routine is a debugging tool which can be used to print the revised entity set table.

#### 4.1.4 TOKENIZER-PROCESSOR

This module serves to tokenize retrieval statements, and to execute "define", "ad hoc", and "listdef" statements. It is the only module of the virtual information facility which communicates with the dictionary of virtual information definitions, besides USER-INTERFACE which makes one call to dictionary for initialization. When tokenizing each retrieval statement, virtual definitions are recalled from the dictionary whenever appropriate and substituted directly into the retrieval statement.

This module contains the following internal routines:

- (1) GETS
- (2) NXTKSTR
- (3) RMVFBKLS (internal to NXTKSTR)
- (4) TOK1 (internal to NXTKSTR)
- (5) DEF
- (6) BDTKCHN
- (7) MSG
- (8) LISTDEF
- (9) DEFDSPLY (internal to LISTDEF)
- (10) REPLACE

The GETS routine is a general tool as described earlier in the USER-INTERFACE module.

The NXTKSTR routine is the core of the tokenizing process; it recognizes from the input stream the next token in the form of a character string. Each token is a separately recognizable entity. This routine is called repeatedly by the BDTKCHN routine which builds an entire chain of tokens.

The RMVFBLS is the same routine as described in the BUFFER module.

The TOK1 routine is the main body of the NXTKSTR routine. It recognizes the next portion of the input string, which is to be transformed into a separate token.

The DEF routine serves to execute the "define" and "ad hoc" data base statements. It creates and modifies virtual information definitions in the dictionary of virtual definitions.

The BDTKCHN routine builds an entire chain of linked tokens. Each retrieval statement is transformed to such a chain of separate tokens before further processing.

The MSG routine outputs a message line to the terminal and prompts the user to press the "enter" key to continue.

The LISTDEF routine executes "listdef" data base statements. It would recall the definition in the dictionary which is to be listed, and output the definition to the terminal.

The DEFDSPLY routine is internal to the LISTDEF routine. It serves to process a stored definition for terminal display. Retranslation is needed to reconstruct those original characters which have been previously translated.

The REPLACE routine is a general tool as described in the BUFFER module.

#### 4.1.5 LANGUAGE DESIGN and SPECIFICATION

This module incorporates the design and formal specification of a data base language which defines and retrieves virtual information. Chapter 5 is devoted exclusively to explaining and describing this module.

#### 4.1.6 FINITE STATE AUTOMATON (MACHINE)

This module serves to parse the data base statements written in the language illustrated in Chapter 5. It consists of a set of Match-Action-Nextstate rules which is one of the inputs to a generalized parse program written by Peter Lu. A change in the grammar of our data base language readily corresponds to



changes in the rules of this finite state machine; thus, freeing us from changing the parse program itself. Chapter 6 is devoted exclusively to explain the workings of these rules.

#### 4.2.0 INTERNAL GLOBAL VARIABLES

##### USER-INTERFACE MODULE:

```
execbuff  -- execution buffer
trnsbuff  -- transaction buffer
firstlast -- passed to BUFFER and used to indicate when
           to terminate end of session.
line      -- used to hold user-input line
```

##### BUFFER MODULE

```
execbuff, trnsbuff, firstlast (same as in USER-INTERFACE)
prstline  -- current input line, char(80)
strnsp    -- prstline, stripped of leading and ending spaces
cplnvar   -- strnsp, char(80) varying
ldspaces  -- number of leading spaces on input line
key       -- current input word to be investigated
```

##### ACTIVITY-COORDINATOR MODULE

DICTIONARY -- virtual information dictionary  
 ENTITY -- entity set representation  
 TOKEN -- token representation  
 MACH -- finite state machine representation  
 XTREE -- execution tree representation  
 XCHANGE -- entity set table representation  
  
 UNIT -- current data base statement to be processed  
 TKLSPTR -- pointer to the list of input tokens  
 GO -- indicator to proceed with beyond the  
           tokenizer-PROCESSOR stage

TOKENIZER-PROCESSOR MODULE:

unit -- current data base statement  
 tklsptr -- pointer to list of input tokens  
 diction -- dictionary  
 go -- indicator to continue processing  
       (set only for retrieval statements)  
 kind -- numeric indicator for arithmetic and  
       string constants.  
 word -- first word of data base statement

#### 5.0.0 LANGUAGE ILLUSTRATION AND SPECIFICATION

This section contains an illustration and a formal specification of the data base language implemented on the virtual information facility, as well as the buffer commands which are implemented to provide an interactive environment in which virtual information processing may be continued.

#### 5.1.0 DATA BASE STATEMENTS

##### 5.1.1 DEFINE STATEMENTS

A user may define the character string *x* to be a macro definition of the character string *y* by the following statement:

```
define x as y ;
```

```
def x as y ;
```

For instance:

```
define currentyear as birthyear + age ;
```

```
define old as age > 60 ;
```

```
define employee as {worker} ;
```

```
def a as 2+3+4/5*8 ;
```

```
def inches as 2.54 * centimeters ;
```

Using define statements to simulate functions:

```
define sum#a#b as #a + #b ;
```

This specifies a function with two arguments, #a and #b.  
The value of sum#2#4 when evaluated would be 6.

To remove previously defined definitions:

```
define age remove ;  
define age rem ;  
define a remove ;
```

### 5.1.2 ADHOC STATEMENTS

Adhoc statements are similar to defines statements; the only difference lie in the target catalogue identity. Adhoc statements operate on the adhoc dictionary, and define statements operate on the permanent dictionary.

```
adhoc x as y ;  
  adhoc curentyear as 1982 ;  
  adhoc sgfhg as kruilko ;  
  adhoc age rem ;  
  adhoc avg#x#y#z as (#x + #y + #z) / 3 ;
```

In both define and adhoc statements, virtual definition may be defined on top of other virtual definitions; in our implementation, we allow a maximum of 10 nested levels of virtual information definition. Thus, if one defines a recursive definition, our system would terminate the entire process of replacing definition names by their associated definitions by the eleventh attempt in replacing

the same definition name.

### 5.1.3 LISTDEF STATEMENTS

These statements list the stored definitions in the dictionary by name; the search order is:

adhoc --> permanent.

```
listdef age ;
```

```
listdef employee ;
```

```
listdef sgfhg ;
```

### 5.1.4 RETRIEVE STATEMENTS

Our retrieve statements are powerful enough to retrieve the following kinds of virtual information from either real or virtual entity sets:

- a) computed facts
- b) implied facts
- c) factored facts

Computed facts are those information derived from an algorithmic computation on existing data; implied facts are those information derived from indirect associations; factored facts are those instances of a particular group of facts which share a certain condition on their attribute values.

We support the following kinds of computational

operators with four levels of precedence, left to right within each level of precedence, and together with parenthesized precedence capability:

+ , - ,   ,	---	lowest order of precedence (plus, minus, and concatenate) (binary infix operators)
* , / ,	---	next order of precedence (multiplication and division) (binary infix operators)
!	---	next order of precedence (exponentiation operator)
+ , - ,	---	highest order of precedence (arithmetic pre-operators)

Aside from these built-in operators, we also support a number of built-in functions as enumerated below:

functions with no arguments:

date            usage --> nextdate = STR (DATE , 2:'\$') + 1 ;  
first, one would use the STRING function to obtain the relevant portion of the value returned by the DATE function, and then this value is incremented by 1.

A date in the system may be stored in the form month\$date\$year, or any other pre-determined

manner. The STRING function is very much suited for the getting of relevant portions of data stored in this form.

Functions with one argument:

These functions operate on entire entity sets; in this sense, they are vertical operators, not of the unilateral kind which we are usually familiar with.

Any valid expression may serve as an argument to built-in functions.

MAX(y)            usage --> max (length + width + height)  
                  refers to that particular instance of the  
                  entity set whose dimensions have a greater  
                  sum than all other members of the set.

retrieve ( { employee} where ( salary = max (salary) ) )  
          by ( {v0} name ) ;  
          gets the name of the employee who earns  
          the highest salary.

SUM(y)            usage --> where ( sum ( y ) = 100 )  
                  yields true if the sum of all instances  
                  of y in the current entity set equals 100.

MIN(a+b-5)

for each member of the set, the value of  
argument expression is first calculated,

then the minimum of them all is taken.

ABS(x+y+z)

returns the absolute value of the argument  
expression for each member of the entity set.

SGN(index)            --        returns -1 if argument is negative  
                                  returns 0    if argument is zero  
                                  returns 1 if argument is positive

SUM(x!2)

sums up the squares of the variable x, yielding  
one single value.

POS(v)                --        returns boolean value for each  
                                  instance of attribute v in the  
                                  entity set.

ZER(x)                --        returns boolean value for each  
                                  instance of attribute x in the  
                                  entity set.

Functions of more than one argument:

STR ( b ,nth occurrence of 'x', mth occurrence of 'y' )  
returns a substring of b from the nth occurrence of 'x'  
to the mth occurrence of 'y' exclusively.  
usage --> STR ( b , 4:'x' , 5:'y' )



a retrieve statement is of the following basic form:

```
retrieve ( list of real and/or virtual entity sets
           separated by commas and each with an optional
           predicate clause )
        by ( entity set designation
            list of items to be retrieved ) ;
```

The first set in the list would be known as {v0}

The second set in the list would be known as {v1}

" " " " " " " " " "

The tenth set in the list would be known as {v9}

A maximum of ten such sets on this level is permitted.

We hope to demonstrate the functionality of the  
retrieve statement through the following examples:

```
retrieve ( {es1} ) by ( {v0} x ) ;
    gets all those "x" attributes of entity set "es1" .
```

```
retrieve ( {es1} ) by ( {v0} x+3 ) ;
    computes and returns all x+3 instances of entity
    set ES1 which has the attribute X.
```

```
retrieve ( {es1},{es2} ) by ( {v0} max (x) )
                        by ( {v1} y*4, min (y*z) ) ;
```

First, it gets the instance of "es1" 's attribute "x"  
which has the highest value of all instances of "es1" 's  
attribute "x",

Second, it gets the "y\*4" elements of entity set "es2", y being an attribute of "es2" , then it gets the instance of "es2" 's "y\*z" which has the minimal value of all other instances of "es2" 's "y\*z", "y" and "z", both being attributes of "es2" .

```
retrieve ( {es1} where ( ( ( x1 < x2 ) and ( x3 = x4 ) ) ) ,
           {es2} where ( y1 = ( 1,2,3,4,5 ) or y2 = y3 ) ,
           {{v0}} where ( x1 | x3 = str ( b,4:'$',5:'$' )))
by ( {v0} x1, x3)
by ( {v1} y1, y2) ;
```

A complete set of predicate conditions on real and virtual entity sets is supported with "and" , "or" , "xor" and "-" connectors, with "<" , ">" , "=" , "!=" , "<" , and ">" relators. the default order of precedence is from left to right unless otherwise indicated by the use of parentheses.

For instance: the following are equivalent conditions:

```
x1 = x2 and x2 > x3 or x3 < x4
(x1 = x2) and (x2 > x3) or (x3 < x4)
((x1 = x2) and (x2 > x3)) or x3 < x4
((((x1 = x2)))) and x2 > x3 or (x3 < x4)
```

Each where clause is attached to the entity set specified immediately prior to the clause itself; the

only restriction on the kinds of entity sets allowed to have where clauses attached to them is that they are not one of the following:

{v0},{v1},{v2},{v3},{v4}  
{v5},{v6},{v7},{v8},{v9}

This is so because these entity sets only refer to some other entity set which was already specified. According to this principle, the following entity sets may have associated predicates because they are themselves the specification of new virtual entity sets:

{{v0}},{{v1}},  
.....{{v9}}

The second entity set in the foregoing retrieve statement has an associated predicate which specified  $y_1 = (1,2,3,4,5)$ ; this predicate requires  $y_1$  to be of either one of the constants within the enclosing set of parenthesis. however, when using this kind of comparison, we make the restriction that all values which appear in the enclosing set of parentheses must be either an arithmetic constant or a string constant.

The third condition clause in the foregoing retrieve statement illustrates the use of the string functions; the function call is attempting to return

the substring of b, from the 4th occurrence of the '\$' character to the 5th occurrence of the '\$' character. The predicate would yield true if the results of concatenating x1 and x3 is equal to the return value of that function call.

```
retrieve ( { {es1} mi (x,y,z) {es2} }  
          by ( {v0} weight ) ;
```

This statement retrieves all instances of the weight attribute of the virtual entity set composed of the "multiple union" of real entity sets es1 and es2, based on the common attributes "x", "y", and "z".

Each virtual entity set, enclosed by a set of left and right braces, may itself be composed of two other virtual entity sets as the result of a set operation, and each of these two component entity sets may also be composed of two other virtual entity sets as the result of a set operation, and each of these component entity sets so on. In this manner, virtual entity sets may be built very quickly one on top of another, each with its own set of predicate conditions to be met.

Five set operators are supported between two entity sets: they are, multiple union, multiple intersection, single union, single intersection, and cartesian product; namely, MU, MI, SU, SI, and CS. The semantics of these

operators are described in the co-thesis by Peter Lu.

```
usage --> {{es1} SI (x) {es2}}  
          {{es1} su (y,z) {es2}}  
          {{es1} MU (y,z,z1) {es2}}
```

The operands of MU, MI, SU, and SI can be a list of attribute names separated by commas, but the operands to CS must be two in number and the first one must be preceded by a " " sign to indicate its cartesianess.

```
{{es1} cs (id,class) {es3}}
```

An arbitrary WHERE clause representing a predicate condition may follow each and every kind of prescribed virtual entity set.

```
{{es1} where (color = 'red') cs (id,class)  
{es3} where (num > 7) } where (size < 5)
```

#### 5.2.0 BUFFER COMMANDS

These interactive commands may be issued by the user via a terminal session with the virtual information facility. They are the means by which an interactive environment is constructed in which the data base commands may be executed.

The buffer is divided into an execution and a transaction buffer. an adhoc dictionary is built for the duration of each transaction in which many data base statements may be strung together and executed sequentially. Thus,

within a transaction a user may operate on either the permanent dictionary shared by itself and any other transaction executed before or after it, or the adhoc dictionary which is for its own exclusive use.

A completed data base statement in the execution buffer will automatically trigger the execution of that statement; therefore, the execution buffer is not suited for the stringing together of multiple statements.

Each buffer command may be entered from within either the transaction buffer or the execution buffer, and may be recognized by two or more initial characters of its full name. furthermore, the contents of the execution buffer and at least 10 lines of the transaction buffer will always be displayed on the terminal.

#### 5.2.1 COMMAND SYNTAX

##### (1) FINPUT 1starg

This command will read the contents of the cms file whose file name is "file", and file type is whatever is entered as "1starg", into the transaction buffer. The original content of the transaction buffer before the execution of this command will be erased.

##### (2) FSAVE 1starg

This command will write into the cms file whose file name is "file", and file type is whatever is entered as "lstarg", from the contents of the transaction buffer. Upon completion of the command, transaction buffer content will be empty.

(3) TRANSACT

This command lets the user enter the transaction buffer.

(4) ENDTRANS

This command lets the user terminate the transaction buffer and enter the execution buffer.

(5) TERMINATE

This command terminates the virtual information facility and returns control to cms.

(6) RUNTRANS

This command executes the contents of the transaction buffer statement by statement.

(7) DODELETE

This command does the same as "runtrans" except that upon its completion, the transaction buffer contents will be erased.

(8) ERASETRANS

This command erases the contents of the transaction buffer.

(9) KILLEXEC

This command erases the contents of the execution buffer.

(10) HELP

This command gives a brief description of all buffer commands.

(11) INSERT 1starg 2ndarg

This command would insert a line of text into the transaction buffer. the first argument is a destination of the line number within the buffer after which the inserted line is to be inserted, and the second argument is the text to be inserted.

(12) DELETE 1starg

This command deletes a line from the current transaction buffer, and the exact line number is specified by the first argument.

(13) TOPLINE 1starg

This command specifies the starting line number of the ten transaction buffer lines which are always displayed, and that number is designated by the first argument.

### 5.3.0 FORMAL BNF DESCRIPTION OF DATA BASE STATEMENTS

\*\*\*\*\*

<defstmt>::= "DEFINE" | "DEF" name defopt ;



```

<defopt> ::= < "AS" <***> > |
           < "REMOVE" | "REM" >

<ad hoc stmt> ::= "ADHOC" name defopt ;

<list stmt> ::= "LISTDEF" name ;

*****

<retr stmt> ::= "RETRIEVE" ( vsets ) byspec ;

<vsets> ::= < "{" vsets1 "}" { "where" ( cond ) } > |
           < "{" vindrs "}" >
           { , vsets }

<vsets1> ::= name |
            combsets

<combsets> ::= < "{" vsets1 "}" { "where" ( cond ) } > |
              < "{" vindrs "}" >
              { setop vsets2 }

<vsets2> ::= < "{" vsets1 "}" { "where" ( cond ) } > |
            < "{" vindrs "}" >

<vindrs> ::= < "V0" | "V1" | "V2" | "V3" | "V4" | "V5"
              | "V6" | "V7" | "V8" | "V9" >
              | < "{" vindrs "}" >

.

<setop> ::= <cs> | <ncs>

<ncs> ::= < "MI" | "SI" | "MU" | "SU" >
          ( reflist )

<reflist> ::= refl { , refl }

<cs> ::= "CS" ( varef , varef )

*****

```

```

<exp>      ::= <(exp)> | exp-inf1 | exp-inf2 | exp-pre
               | exp-pwr | exp-prim

<exp-inf1>  ::= exp inf1-op <(exp)> | exp-inf2 | exp-prim
               | exp-pwr

<exp-inf2>  ::= exp2 inf2-op <(exp)> | exp-prim | exp=pwr

<exp2>      ::= <(exp2)> | exp-inf2 | exp-pre
               | exp-pwr | exp-prim

<inf1-op>::= + | - | "|"
<inf2-op>::= * | /

<exp-pre>::= pre-op < exp-prim | exp-pre | exp-pwr >
<pre-op> ::= + | -

<exp-prim>  ::= ref | const

<exp-pwr>::= exp-prim ! < exp-prim | exp-pre >

<ref>      ::= refl | funref
<refl>     ::= { } varef
<varef>    ::= name { ( varef ) }
<const>    ::= fixed | integer
<fixed>    ::= integer . { integer }
<digit>    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer>::= digit | digit integer
*****

<funref> ::= singfun | strfun
<singfun>::= funame ( exp )
<funame> ::= "MAX" | "MIN" | "ABS" | "POS"
               | "SGN" | "ZER" | "SUM"

<strfun> ::= "str" ( strarg )

```

<strarg> ::= exp , exp { : exp } strarg1

<strarg1> ::= { < @ exp { : exp } > | < , exp { : exp } }

\*\*\*\*\*

<cond> ::= < ( cond ) > | cond1

<cond1> ::= smpcnd | < cond condop cond2 >

<cond2> ::= smpcnd | < ( cond ) >

<smpcnd> ::= < ( smpcnd ) > | < { not } ( smpcnd ) >  
| < exp relop exp >

<not> ::= ~ | <> not

<condop> ::= "AND" | "OR" | "XOR"

<relop> ::= = | > | <  
| { not } =  
| { not } >  
| { not } <

\*\*\*\*\*

name            any character string of length less than 17  
                 and composed only of the following characters:  
                 abcdefghijklmnopqrstuvwxyz\$#&\_? "

\*\*\*\*\*

### 5.3.1 BNF SUPPLEMENT

- \*    comments are enclosed within "\" characters
- \*    quote characters within string constants are represented  
     by two consecutive single quote characters
- \*    string constants are enclosed by "'", single quote

characters

- \* <\*\*\*> designates any arbitrary character string
- \* single line comments enclosed by "\"" characters are permitted before, after, and within each data base statement, as well as before and after each buffer command line. They are eventually removed, and are not recognized as part of any input line.
- \* the system makes no distinction between lower and upper case characters.

\*\*\*\*\*

#### 6.0.0 FINITE-STATE-MACHINE (PUSH-DOWN-AUTOMATON)

In this chapter, the Finite-State-Machine used to parse data base retrieval statements would be briefly described. A Finite-State-Machine consists of a number of states, one of which is a start state, and one or many of which may be an ending state. There also is a pointer which would point to the current word being examined on the user given statement being processed. In our case, the user input is always a retrieval statement written in the data base language presented in Chapter 5, and each word would always be a single token along the token chain to which the original retrieval statement has already been transformed to.

Each state within the machine has a set of match-next\_state rules, and collectively the union of these sets of rules regulate the behavior of the finite-state machine on any given input. Each state attempts to find a match between the current word and the match section of any one of its rules, and if a match is found, then control is passed to the state identified by the next\_state section of the matching rule, and the input pointer points to the next word of the statement being processed. If the end of input is ever reached, and control happens to be within an ending state, then the machine halts and is said to have accepted the statement which it had just processed.

Our construction of such a finite-state-machine went straight on to meet a number of problems. First, such a machine has no provisions for any processing except for moving from state to state. Thus, we augmented our design to a Push-Down-Automaton which is a finite-state-machine with auxiliary memory and data movement capabilities. In fact, in order to generate an execution tree and various tables from a given retrieval statement, we had to augment the processing ability of the automaton by a set of action routines, and transform the format of state-rules to a tri-tuple consisting of a match section, an action section, and a next\_state section.

The auxiliary memory we have chosen for the automaton is in the form of two stacks, an operator stack and an operand stack. The match section of each rule has provisions to match either the current word on three different sources, the input token chain, the top of stack #1, and the top of stack #2. Actually, when the source is the input token chain, an added ability to match for a given class of tokens as well as a specific token is available. The action section of each rule has provisions for pushing and popping the current input token, onto or off either stack #1 or stack #2, and for invoking other action routines which generates an execution tree and various tables from the current elements on both stacks, and modifies the current contents of the stacks. The next\_state section of each rule contains the state ID number which identifies the next state to

which control would be passed to after the proper action routines in the matching rule have been executed. Our machine is deterministic in nature; by this we mean that for a given combination of input token, top of stack #1, and stack #2, there is at most one next\_state from which control may go to after the current state. A non-deterministic machine would have been condensed, but also more complex and harder to implement because of the need to backtrack over decision points.

The construction of this push-down-automaton is logically divided into two parts, the writing of the match-action-next\_state rules, and the writing of a program which takes these rules as an input and sets up the proper environment in which data would be matched, actions would be performed, and next\_states would be go to, exactly according to the specifications of the prescribed match-action-next\_state rules. The front-end of the virtual information facility, as presented in this thesis, is responsible for the writing of these match-action-next-state rules, and the implementation of these rules is a responsibility of the back-end.

#### 6.1.0 CONFIGURATION

Action routine implementations are part of the back-end written by Peter Lu in his concurrent thesis. These programs are within the PARSE module as illustrated in figure 4.1 . As

part of the front-end, the match-action-next\_state rules are within the FINITE-STATE-MACHINE module and currently reside in CMS file "file machin" . A DEFMCH module is written to establish the machine environment, taking the contents of "file machin" as input, and the PARSE module, when called upon, activates the finite-state-machine.

#### 6.2.0 MATCH-ACTION-NEXT\_STATE RULES

Match-Action-Next\_State rules is a 3-tuple of information. The first component prescribes what to match for a certain element on either one or more than one of the following sources, the input stream, top element of stack #1, and top element of stack #2. The second component is a sequence of action routine invocations; these routines are to be executed whenever the matching component of the same rule matches. The third component is a state-number representing the next state to which control is to go when the action routines in the same rule have been executed.

The three components of each rule is separated from each other by the "\" character as illustrated in the following:

\ match component \ action component \ next\_state number \



Furthermore, since these rules prescribe the transitions from state to state, they are referred to as the "transition rules", and the full specification of a transition rule is as follows:

t \ match component \ action component \ next\_state number \

The specification of a state is accomplished first by writing the following to indicate the identity of the state:

s \ state number \

and then by a listing of the match-action-next\_state rules which belong to this particular state. The sequential order of rules in this list can not be inter-changed, because when control comes to each state, the rules will be tried sequentially in the order of their position on the list. Thus, a sample state specification is as the following:

s \ 25 \  
t \ retrieve \ del \ 26 \  
t \ ( \ pop, 1 \ 27 \  
t \ + \ pop, 2 \ 36 \

The foregoing rules would first try to match the word from the input, if it is matched, then the action routine "del",

delete input token, would be executed, and then control would go to state number 26. If the first rule did not match, then the second rule which attempts to match a "(" character on the input would be tried. If this rule matches, then the "pop" routine would be executed, and stack #1 would be popped, and control would go to state number 27. If the second rule did not match, then the machine would try the third rule, which matches for the "+" operator on the input stream, if it is matched, then stack #2 would be popped and control would go to state number 36. If none of the rules for a the current state matches, then the machine would signal premature termination, which means that the input is invalid, and diagnostic messages would be sent to CMS file "file error". State number 0 is the final state of the machine, and if control is ever passed to this state, then the input is valid, accepted, and successfully processed; the associated execution tree and entity set tables would have already been generated and available for use by the following stages of the virtual information facility.

The match component of each rule is composed of zero to three separate parts, each of which is separated from the other by a "," character. The first part represents the input source, the second part represents the source from stack #1, and the third part represents the source from stack #2. If non of the three parts exist, then that particular rule would match everything and anything, and the corresponding action routines would

always be executed if the machine ever tries to match that rule.

For instance:

s\1\

t\ a,b,c\del\2\

The foregoing rule would match simultaneously a character "a" on the input stream, a character "b" from the top of stack #1, and a "c" character from the top of stack #2. All three sources must be matched before the corresponding action routines may be executed. If any of the sources does not match, then this entire rule is not matched, and either the next rule in the sequence would be matched or the machine would signal premature termination if there are no more rules to be matched for this state. Thus, at most three sources may be matched in the match component of the rule, and at most one single token may be matched on any one source.

The action component of each rule may contain calls to more than one action routine. These action routine invocations are written in sequential order and are separated by the "|" character; these routines would be executed in the order of their appearance in the action component. For instance:

s\5\

```
t\ + \ push,2,i@ | del | pop,1 \ 6 \
```

The foregoing rule would match the "+" character on the input stream, and then execute the three action routines in sequential order. First, it would push the "+" character on to stack #2, as specified by the first routine call, then it would delete the current character on the input stream, thereby advancing the input pointer to point to the next input token, and then it would pop stack #1, popping off stack #1's top element.

In order to facilitate the matching of a group of symbols, not necessarily all of the same classification, we have developed the concept of a "cluster"; a cluster simply is a union of one or more prescribed symbols which may be matched under one cluster name. All cluster names begin with a "@" character, and they provide an added convenience for the making of transition rules. For instance, an arithmetic cluster may include the + and - characters, and be named "@sumop". By using the name @sumop in the match component, we may match either the + or the - characters. We currently have the following groups of clusters:

```
@sumop    ---    + , -  
@sumop>   ---    + , - , * , / , !  
@multop   ---    * | /
```

```

@multop> ---  * , / , !
@conc    ---  |
@conc>   ---  | , + , - , * , / , !
@virt    ---  v0 , v1 , v2 , v3 , v4 ,
           v5 , v6 , v7 , v8 , v9
@rel     ---  > , <
@cmp     ---  and , or , xor
@setop   ---  cs , mu , mi , su , si

```

It was mentioned earlier that a rule may match from the input source a token of a specific classification; to do this, the rule indicates the class of characters it would be matching for by writing a ":" character and followed by the class representation character. There are altogether eight different classes of tokens, namely the classes A , N , D , O , B , Q , M , and S .

Characters belonging to class A are the following:

abcdefghijklmnopqrstuvwxyz

Characters belonging to class N are the following:

0123456789

Characters belonging to class D are the following:

.

Characters belonging to class O are the following:

+ - / \* ! |

Characters belonging to class B are the following:

- = > < , ( ) ° { }

Characters belonging to class Q are the following:

'

Characters belonging to class M are the following:

" ? ¢ \_ & ± # \$

Characters belonging to class S are the following:

%

The following rule matches for a token of class M on the input stream, does nothing, and then passes control to state 4.

s \ 3 \

t \ : M \ \ 4 \

The parsing of any language may sometimes be facilitated by the creation of sub-parsers which parse a subset of the language. The usefulness of this idea is demonstrated by our using of the sub-routine concept in the finite-state-machine. Two sub-machines were written, one to parse expressions, and one to parse entity set conditions which calls on the

expression parser. The idea is to pass control to the sub-parser, and leave subroutine return command and address on top of stack #2 before entering the sub-parser. When the sub-parser finds a negative state number as the next\_state, it should have that return command and address available on top of stack #2, and should pop that element off stack #2, and then pass control to the state identified by that command. For example, the following rule passes control to a sub-parser which starts on state 30, and also specifies the return state number as 80 when the sub-parser finds a negative-state number in the next\_state component. With this strategy, the last rule which indicate the successful parse of the sub-parser must have a negative state number in the next\_state component.

```
s \ 20 \
t \ d \ del | push,2,subr:80 \ 30 \
```

### 6.3.0 ACTION ROUTINES

The following action routines are written within the back-end of the facility and are available for use in the action component of the match-action-next\_state rules:

Routine	Usage	Semantics
---------	-------	-----------

POP	pop,1	pops stack #1
	pop,2	pops stack #2
PUSH or P	push,1,i@	pushes the input
	p,1,i@	token onto stack #1
	push,1,c@	pushes the input token
		concatenated by a ":"
		and its classification
		onto stack #1
	push,2,i@:2	pushes the input token
		concatenated by ":2"
		onto stack #2.
		Frequently, this is used
		to associate the expected
		number of operands to the
		operator which is being
		pushed onto the operator
		stack.
DEL	del	advances the input pointer
		to point to the next input
		token.



Non-stack related routines:

GENNODE or GD

generates an operator node with its specified number of children which are found on stack #1 as a partial execution tree. The address to the partial tree just generated is placed back on top of stack #1.

INDX

adds the first level of indirection to a data element

ADDON

adds an additional level of indirection

VIRTX

exchanges the addresss of the indicated virtual entity set

MULX

generates a multiple "OR" node

#### ATTWHR

attaches the address of a condition to  
its associated virtual entity set

#### GENENT

generates a new virtual entity set

#### VIRTA

adds the cuurent virtual entity set to  
an entity set table

The non-stack relate routines listed above have much to do with the internal workings of the back-end and are left to be more precisely explained by the back-end documentation.

#### 6.4.0 LISTING

The rules written in CMS file "file machin" are not readily readable because of its syntax; a FORMMCH program is available to format it to a readable form as shown in a listing of the rules on the following pages:

# FINITE-STATE-MACHINE RULES MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 1  
1. 1 RETRIEVE

STATE NUMBER: 2  
2. 1 :..RETRIEVE  
2. 2 BY

STATE NUMBER: 3

STATE NUMBER: 11

11. 1 -  
11. 2 ( :A: :S  
11. 3 :A: :S  
11. 4 -  
11. 5 +  
11. 6 :R  
11. 7 STR  
11. 8 DATE  
11. 9 :B

STATE NUMBER: 12

12. 1 )..(  
12. 2 :..P  
12. 3 :  
12. 4 @SUMOP..@SUMOP>  
12. 5 @SUMOP>..@CONC  
12. 6 @MULTOP..@MULTOP>  
12. 7 @MULTOP..@SUMOP  
12. 8 @CONC..@CONC>  
12. 9 @CONC>  
12. 10 :..SUBR  
12. 11

STATE NUMBER: 13

13. 1 :R  
13. 2 :A: :S  
13. 3 -  
13. 4 +  
13. 5 ( :  
13. 6 :B

STATE NUMBER: 14

14. 1 ( :  
14. 2

STATE NUMBER: 15

15. 1 :R

STATE NUMBER: 16

16. 1 -  
16. 2 +  
16. 3 :A: :S  
16. 4 :R

47

PUSH. 2. 1@DEL; PUSH. 2. SUBR: 2

0

140

DEL; PUSH. 2. SUBR: 2

18

12

12

16

11

14

22

12

20

DEL  
PUSH. 2. 1@DEL  
PUSH. 1. @DEL  
PUSH. 2. 1@: 1; DEL  
DEL  
PUSH. 1. @DEL  
PUSH. 2. SUBR: 12; PUSH. 2. 1@: 6; DEL  
PUSH. 2. 1@: 0; GENNODE; DEL  
PUSH. 2. SUBR: 12; PUSH. 2. 1@: 1; DEL

12

12

13

12

13

12

13

12

13

-1

12

DEL; POP. 2  
GENNODE  
PUSH. 2. 1@: 2; DEL  
GENNODE  
PUSH. 2. 1@: 2; DEL  
GENNODE  
PUSH. 2. 1@: 2; DEL  
GENNODE  
PUSH. 2. 1@: 2; DEL  
GENNODE

14

12

16

13

13

17

PUSH. 1. @DEL  
PUSH. 1. @DEL  
PUSH. 2. 1@: 1; DEL  
DEL  
PUSH. 2. 1@DEL  
PUSH. 2. 1@: 1; DEL

15

12

PUSH. 2. 1@DEL; INDX. 1

19

PUSH. 1. 1@DEL

16

16

12

14

PUSH. 2. 1@: 1; DEL  
DEL  
PUSH. 1. @DEL  
PUSH. 1. @DEL

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 17			
17. 1 (	PUSH, 2, 1, 0, DEL	11	
STATE NUMBER: 18			
18. 1 : R	PUSH, 1, 0, : C, DEL	14	
STATE NUMBER: 19			
19. 1 (	PUSH, 2, 1, 0, DEL, ADDON, 1, 0, POP, 1	15	
19. 2 ) . . (	ADDON, 1, 0, POP, 1, DEL, POP, 2	28	
STATE NUMBER: 20			
20. 1 (	DEL, PUSH, 2, SUBR: 21	11	
STATE NUMBER: 21			
21. 1 : . SUBR	DEL, GENNODE	-1	
21. 2 )		21	
STATE NUMBER: 22			
22. 1 (	DEL, PUSH, 2, SUBR: 23	11	
STATE NUMBER: 23			
23. 1 : . SUBR	DEL, PUSH, 2, SUBR: 24	-1	
23. 2 % 0		11	
STATE NUMBER: 24			
24. 1 : . SUBR	DEL, P, 1, : -1, P, 1, : -1, P, 1, : -1, P, 1, : -1, GD	-1	
24. 2 )	DEL, P, 1, : -1, P, 1, : -2, P, 2, SUBR: 26	24	
24. 3 % 0	DEL, P, 1, : -1, P, 1, : -3, P, 2, SUBR: 26	11	
24. 4 % 5	DEL, P, 1, : -1, P, 1, : -3, P, 2, SUBR: 26	11	
24. 5 % 3	DEL, P, 2, SUBR: 25	11	
STATE NUMBER: 25			
25. 1 : . SUBR	DEL, P, 1, : -1, P, 1, : -1, P, 1, : -1, GD	-1	
25. 2 )	DEL, P, 1, : -2, PUSH, 2, SUBR: 26	25	
25. 3 % 0	DEL, P, 1, : -3, PUSH, 2, SUBR: 26	11	
25. 4 % 5		11	
STATE NUMBER: 26			
26. 1 : . SUBR	DEL, P, 1, : -1, GENNODE	-1	
26. 2 )	DEL, PUSH, 2, SUBR: 27	26	
26. 3 % 3		11	
STATE NUMBER: 27			
27. 1 : . SUBR	DEL, GENNODE	-1	
27. 2 )		27	
STATE NUMBER: 28			
28. 1 ) . . (	DEL, POP, 2	28	
28. 2		12	
STATE NUMBER: 29			

STATE NUMBER: 30	30. 1 (	PUSH, 2, I, I	31
STATE NUMBER: 31	31. 1 ^	PUSH, 2, I, I	36
31. 2 (	PUSH, 2, I, I	32	
31. 3	PUSH, 2, SUBR: 33	11	
STATE NUMBER: 32	32. 1 (	PUSH, 2, I, I	32
32. 2	PUSH, 2, SUBR: 33	11	
STATE NUMBER: 33	33. 1 ^	PUSH, 2, I, I	37
33. 2 ) .. (	POP, 2	33	
33. 3 *	PUSH, 2, I, I	38	
33. 4 REL	PUSH, 2, I, I	34	
STATE NUMBER: 34	34. 1 (	PUSH, 2, I, I	34
34. 2	PUSH, 2, SUBR: 35	11	
STATE NUMBER: 35	35. 1 ) .. (	POP, 2	35
35. 2 .. REL	GENNODE	35	
35. 3 ^	GENNODE	35	
35. 4 .. CMP	GENNODE	35	
35. 5 CMP	PUSH, 2, I, I	31	
35. 6 .. SUBR		- 1	
STATE NUMBER: 36	36. 1 ^	PUSH, 2, I, I	36
36. 2 (	PUSH, 2, I, I	32	
STATE NUMBER: 37	37. 1 ^	PUSH, 2, I, I	37
37. 2 REL	PUSH, 2, I, I	34	
STATE NUMBER: 38	38. 1 (	PUSH, 2, I, I	39
38. 2		34	
STATE NUMBER: 39	39. 1 : A : S	PUSH, 1, I, I	40
39. 2		34	
STATE NUMBER: 40	40. 1 ) .. (	DEL, POP, 2	35
40. 2 %O	DEL, MULX, 1		41

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 41			
41. 1 : A ; S	PUSH, 1, 1, 0, DEL		42
STATE NUMBER: 42			
42. 1 ) ; . (	DEL, POP, 2, ADDON, 1, 0, POP, 1, GENNODE		35
42. 2 %O	DEL, ADDON, 1, 0, POP, 1		41
STATE NUMBER: 43			
STATE NUMBER: 47			
47. 1 (	DEL		48
STATE NUMBER: 48			
48. 1 {	PUSH, 2, 1, 0, DEL		54
STATE NUMBER: 49			
49. 1 ) . 0VIRT	VIRTX, 1, 1, 0, DEL		52
49. 2 )	GENENT, DEL		50
STATE NUMBER: 50			
50. 1 WHERE	DEL, PUSH, 2, SUBR, 51		30
50. 2			52
STATE NUMBER: 51			
51. 1	ATTWHR		52
STATE NUMBER: 52			
52. 1 ;			53
52. 2 BY			53
52. 3 )	DEL		53
52. 4 %O	DEL, VIRTX, PUSH, 2, SUBR, 52		48
STATE NUMBER: 53			
53. 1 , , SUBR	-1		-1
STATE NUMBER: 54			
54. 1 : R	PUSH, 1, 1, 0, DEL		49
54. 2 0VIRT	PUSH, 1, 1, 0, DEL		49
54. 3 {			100
STATE NUMBER: 55			
55. 1 {	PUSH, 2, 1, 0, DEL		56
STATE NUMBER: 56			
56. 1 : R	PUSH, 1, 1, 0, DEL		57
56. 2 0VIRT	PUSH, 1, 1, 0, DEL		57
56. 3 {			112

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 57		
57. 1 } @VIRT	VIRTX, 1, 1@ DEL	60
57. 2 }	GENENT DEL	58
STATE NUMBER: 58		
58. 1 WHERE	DEL PUSH, 2, SUBR: 59	30
58. 2		60
STATE NUMBER: 59		
59. 1	ATTWHR	60
STATE NUMBER: 60		
60. 1 @SETOP		70
60. 2 ..SUBR		-1
60. 3 }	GENENT DEL	58
STATE NUMBER: 61		
61. 1 {	PUSH, 2, 1@ DEL	62
STATE NUMBER: 62		
62. 1 :R	PUSH, 1, C@ DEL	63
62. 2 @VIRT	PUSH, 1, 1@ DEL	63
62. 3 {		124
STATE NUMBER: 63		
63. 1 } @VIRT	VIRTX, 1, 1@ DEL	65
63. 2	GENENT DEL	64
STATE NUMBER: 64		
64. 1 WHERE	DEL PUSH, 2, SUBR: 65	30
64. 2		66
STATE NUMBER: 65		
65. 1	ATTWHR	66
STATE NUMBER: 66		
66. 1 ..SUBR	GENENT DEL	-1
66. 2 }		64
STATE NUMBER: 67		

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 70			
70. 1 CS	PUSH, 2, 1, DEL	71	
70. 2		80	
STATE NUMBER: 71			
71. 1 (	PUSH, 2, 1, DEL	72	
STATE NUMBER: 72			
72. 1 -	DEL	73	
STATE NUMBER: 73			
73. 1 :R	PUSH, 1, C, V; C; DEL	74	
STATE NUMBER: 74			
74. 1 (	PUSH, 2, IND( DEL; IND, 1	75	
74. 2		78	
STATE NUMBER: 75			
75. 1 :R	PUSH, 1, 1, DEL	76	
STATE NUMBER: 76			
76. 1 (	PUSH, 2, IND( DEL; ADDN, 1, POP, 1	75	
76. 2 )..IND(	ADDN, 1, POP, 1; DEL; POP, 2	77	
STATE NUMBER: 77			
77. 1 )..IND(	DEL; POP, 2	77	
77. 2		78	
STATE NUMBER: 78			
78. 1 %O	PUSH, 2, 1, DEL	90	
STATE NUMBER: 79			
STATE NUMBER: 80			
80. 1	PUSH, 2, 1, DEL	81	
STATE NUMBER: 81			
81. 1 (	PUSH, 2, 1, DEL	82	
STATE NUMBER: 82			
82. 1 :R	PUSH, 1, C, V; DEL	83	
STATE NUMBER: 83			
83. 1 )	PUSH, 2, 1, DEL	61	
83. 2 %O	PUSH, 2, 1, DEL	82	
83. 3 (	PUSH, 2, IND( DEL; IND, 1	84	
STATE NUMBER: 84			
84. 1 :R	PUSH, 1, 1, DEL	85	



FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 85			
85. 1 (			86
85. 2 )..IND(	PUSH, 2, IND( {DEL, ADDON, 1e} POP, 1		86
	ADDON, 1e} POP, 1} DEL} POP, 2		
STATE NUMBER: 86			
86. 1 )..IND(			86
86. 2	DEL} POP, 2		87
STATE NUMBER: 87			
87. 1 )			61
87. 2 %O	PUSH, 2, 1e} DEL		82
	PUSH, 2, 1e} DEL		
STATE NUMBER: 88			
STATE NUMBER: 90			
90. 1 :R	PUSH, 1, Ce; V: } DEL		91
STATE NUMBER: 91			
91. 1 (			92
91. 2	PUSH, 2, IND( {DEL} INDX, 1		94
STATE NUMBER: 92			
92. 1 :R	PUSH, 1, Ce; V: } DEL		93
STATE NUMBER: 93			
93. 1 (			92
93. 2 )..IND(	PUSH, 2, IND( {DEL, ADDON, 1e} POP, 1		94
	ADDON, 1e} POP, 1} DEL} POP, 2		
STATE NUMBER: 94			
94. 1 )..IND(			94
94. 2 )	DEL} POP, 2		61
	PUSH, 2, 1e} DEL		
STATE NUMBER: 95			
STATE NUMBER: 100			
100. 1 ..(			101
100. 2	PUSH, 2, SUBR: 52		111
STATE NUMBER: 101			
101. 1 ..(			102
101. 2	PUSH, 2, SUBR: 52		110
STATE NUMBER: 102			
102. 1 ..(			103
102. 2	PUSH, 2, SUBR: 52		109
STATE NUMBER: 103			
103. 1 ..(			104
103. 2	PUSH, 2, SUBR: 52		108

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 104			
104. 1 ..{	POP, 2		105
104. 2	PUSH, 2, SUBR: 52		107
STATE NUMBER: 105			
105. 1	PUSH, 2, SUBR: 52		106
STATE NUMBER: 106			
106. 1	PUSH, 2, {		107
STATE NUMBER: 107			
107. 1	PUSH, 2, {		108
STATE NUMBER: 108			
108. 1	PUSH, 2, {		109
STATE NUMBER: 109			
109. 1	PUSH, 2, {		110
STATE NUMBER: 110			
110. 1	PUSH, 2, {		111
STATE NUMBER: 111			
111. 1			55
STATE NUMBER: 112			
112. 1 ..{	POP, 2		113
112. 2	PUSH, 2, SUBR: 60		123
STATE NUMBER: 113			
113. 1 ..{	POP, 2		114
113. 2	PUSH, 2, SUBR: 60		122
STATE NUMBER: 114			
114. 1 ..{	POP, 2		115
114. 2	PUSH, 2, SUBR: 60		121

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 115			
115. 1 .. (	POP, 2		116
115. 2	PUSH, 2, SUBR: 60		120
STATE NUMBER: 116			
116. 1 .. (	POP, 2		117
116. 2	PUSH, 2, SUBR: 60		119
STATE NUMBER: 117			
117. 1	PUSH, 2, SUBR: 60		118
STATE NUMBER: 118			
118. 1	PUSH, 2, (		119
STATE NUMBER: 119			
119. 1	PUSH, 2, (		120
STATE NUMBER: 120			
120. 1	PUSH, 2, (		121
STATE NUMBER: 121			
121. 1	PUSH, 2, (		122
STATE NUMBER: 122			
122. 1	PUSH, 2, (		123
STATE NUMBER: 123			
123. 1			55
STATE NUMBER: 124			
124. 1 .. (	POP, 2		125
124. 2	PUSH, 2, SUBR: 66		135
STATE NUMBER: 125			
125. 1 .. (	POP, 2		126
125. 2	PUSH, 2, SUBR: 66		134
STATE NUMBER: 126			
126. 1 .. (	POP, 2		127
126. 2	PUSH, 2, SUBR: 66		133
STATE NUMBER: 127			
127. 1 .. (	POP, 2		128
127. 2	PUSH, 2, SUBR: 66		132
STATE NUMBER: 128			
128. 1 .. (	POP, 2		129
128. 2	PUSH, 2, SUBR: 66		131

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 129			
129. 1			130
STATE NUMBER: 130			
130. 1	PUSH.2.SUBR:66		131
STATE NUMBER: 131			
131. 1	PUSH.2.(		132
STATE NUMBER: 132			
132. 1	PUSH.2.(		133
STATE NUMBER: 133			
133. 1	PUSH.2.(		134
STATE NUMBER: 134			
134. 1	PUSH.2.(		135
STATE NUMBER: 135			
135. 1	PUSH.2.(		55
STATE NUMBER: 136			
STATE NUMBER: 140			
140. 1 (			141
STATE NUMBER: 141			
141. 1 (	DEL		142
STATE NUMBER: 142			
142. 1 @VIRT	DEL		143
STATE NUMBER: 143			
143. 1 )	PUSH.1.1@DEL		144
STATE NUMBER: 144			
144. 1 )	DEL		53
144. 2	PUSH.2.SUBR:145		11
STATE NUMBER: 145			
145. 1	ATTBY POP.1		146
STATE NUMBER: 146			
146. 1 )	DEL		53
146. 2 %O	DEL PUSH.2.SUBR:145		11

STATE NUMBER: 147  
MACHINE DEFINITION COMPLETE

#### 7.0.0 MAJOR DESIGN ISSUES

The following are some of the decisions which we had to make in the design of the virtual information facility.

##### 7.1.0 FORM OF STORAGE FOR VIRTUAL DEFINITIONS

How can virtual definitions be stored? We had two viable alternatives. One way is to store the definitions just as they are, in the form of character strings, and when in use, the definition would be substituted within the actual data base retrieval statement in place of the virtual definition name. The alternative to this strategy is to parse the definitions ahead of time, generate the associated execution tree and entity set tables, and when in actual use, the partial execution tree would be simply attached to the main execution tree as an extended subtree, and the main entity set tables would simply be augmented to include the partial table built from the definitions.

The method of parsing the definitions first is similar to the process of compilation. When in actual use, previously defined definitions need not be processed again and again. The method of storing the definitions as they are is similar to the process of interpretation. Each time a definition is used, the

entire process of parsing, tree building and table generation would have to be repeated.

Storing definitions as they are enhances the flexibility of virtual information. Definitions may be created, modified, and even deleted with great ease and efficiency. Furthermore, it eliminates the need to rebuild itself when users request a listing of the stored definitions. It also would enable a generalized macro facility in which not only legitimate and coherent definitions may be stored, but also the seemingly illogical and incoherent definitions as well.

Parsing the definitions as soon as they are defined is not an easy task. Many times, without the proper context in which the definitions are to be used, the associated semantics are not always clear. Even if we can get around this problem by restricting the potential contexts in which each definition may be used, we still would be encounter complicated problems in frequent tree manipulation. Re-shaping an execution tree is a very "messy" task, and would be prone to erroneous branch connections; traversing a huge tree is also not a reasonably efficient operation.

Thus, mainly for the foregoing reasons, we have decided on the first strategy, storing them as they are until invocation, to store virtual definitions.

### 7.2.0 PARSER STRUCTURE

Two strategies were given serious consideration for the parsing of data base statements written in the language specified in Chapter 5. One method is to construct a FINITE-STATE-MACHINE which includes a set of match-action-next\_state rules that correspond to the grammar rules of the data base language. In this manner, these match-action-next\_state rules are inputs to the actual parser just like the data base statements which are to be parsed; with this approach, changes in grammar rules readily correspond to changes in the machine rules. The other method is to construct a conventional parser in which grammar rules are part of the parser program itself.

The finite-state-machine strategy has a highly modular characteristic and gives added flexibility to the data base language in terms of modifiability; however, it would be the first of such machines ever written by the author. The decision was made in favor of the finite-state-machine because the definite gains of this approach seem to surpass the potential for failure of its implementation.

### 7.3.0 PROGRAM CONTROL STRUCTURE

A decision was made to implement a centralized and horizontal control structure for the passing of program control from one to another. The alternative is to build a vertical control structure in which modules are nested one within another, and control may propagate many levels deep before suddenly jumping out to the top. The centralized control structure features an activity coordinator to which control must return to from each module before it is passed to another. Although the vertical approach may seem more natural, the horizontal approach is more adapted to the idea of a single virtual information level within the hierarchical design of INFOPLEX. Furthermore, the horizontal approach contributes more to program modularity with its regard for each module as a separate and un-nested entity. For these reasons, the decision was made to build a centralized and horizontal control structure.

#### 7.4.0 INTERACTIVE EDITOR

Consideration was given to the question of whether or not to build an interactive, full-screen editor in real-time, similar to a miniature EMACS or XEDIT editor as part of the user-interface developed for the virtual information facility. The seriousness of the consideration remained questionable to this day. The argument against it is that the buffer program already supports the capability of inputting the transaction buffer content from an arbitrary CMS file; a user of virtual



information may readily use the XEDIT editor available on CMS to edit their transaction stored in a CMS file, and later input that transaction to the transaction buffer through the FINPUT buffer command. Our interactive, full-screen line editor would take some effort to develop, and still would not be nearly as powerful as XEDIT. In other words, resources may be better utilized if spent on other areas of the virtual information facility. The argument for such an editor is simply that it would provide the added flexibility to change modify buffer contents from within the virtual information facility.

Finally, a decision was made to build a primitive line editor with display capabilities. This is a compromise between the two extremes; not much resources in terms of man-hours would be spent building such an editor, and it would give users of virtual information an added flexibility and convenience in being able to edit their transaction buffer content from within the facility.

#### 7.5.0 LANGUAGE DESIGN

A decision was made to support infix arithmetic and string operators instead of operators in prefix or lisp notation. Although infix operators give rise to a language more difficult to parse, they are more user-friendly. Also, a decision was made to support the capability of explicitly over-riding the

AD-A116 962

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA

F/S 9/2

VIRTUAL INFORMATION FACILITY OF THE INFOPLEX SOFTWARE TEST VEHIC--ETC(U)

MAY 82 J LEE

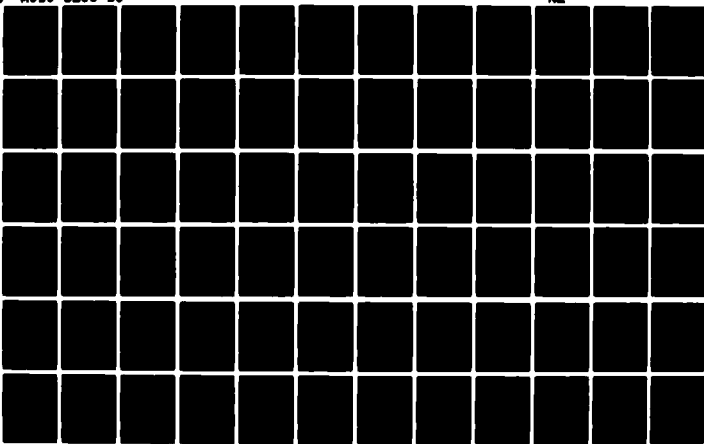
N00039-81-C-0663

ML

UNCLASSIFIED

N010-8205-10

2 of 2  
AD-A  
16502



END  
DATE  
FILMED  
8-82  
DTIC

natural operator precedences by the use of parentheses in arithmetic, string, as well as boolean expressions. This capability makes more difficult the parsing process, but gives much added power and flexibility to the language. In essence, the added advantages of infix operators and use of parentheses for specification of precedence are considered well worth their cost of implementation.

#### 8.0.0 CONCLUSION

Progress was made steadily and swiftly all through the first two months of design and implementation. Then, as precious time passed by each day, increasing hours of work were required for the prompt completion of all thesis objectives. Eventually, all available time was devoted to thesis work and efforts on academic courses became nearly non-existent.

Finally, the complete design and an initial version of the implementation were completed and integrated with Peter Lu's back-end to set up the first virtual information facility in operation on the INFOPLEX software test vehicle, a software simulation of the INFOPLEX data base computer. An extensive set of improvised test cases were written and tested on the facility. The internal interface to the back-end, namely, the instructions issued within the finite-state rules does to conform to expectations. Although the back-end is not yet able to integrate with the lower level of INFOPLEX to access real data, it is able to generate correct information requests based on the execution tree and entity set table established through finite-state-machine instructions which are issued within the front-end.

The results of the implementation give support to the design decisions which were made, especially the decision to con-

struct a finite-state machine and to keep virtual definitions as they are. Most thesis objectives were achieved except for the need of more rigorous test cases to establish the integrity of the facility. Instantly, this facility, when eventually integrated to the next level of INFOPLEX hierarchy, would greatly extend the power and capability of the data base.

## Bibliography

1. Harry R. Lewis, Christos H. Papadimitriou,  
'Elements of the Theory of Computation'  
Prentice-Hall Software Series
2. David Gries,  
'Compiler Construction for Digital Computers'  
John Wiley & Sons
3. Jeffrey Folinus, Stuart E. Madnick, Howard Schutzman,  
'Virtual Information in Data Base Computers'  
Center for Information Systems Research, M.I.T.
4. A Klug, D Tsichritzis,  
Multiple View Support Within the Ansi/Sparc Framework  
Center for Information Systems Research, M.I.T.
5. Meichun Hsu  
'FSTV - The Software Test Vehicle for the Functional  
Hierarchy of the INFOPLEX Data Base Computer'  
Center for Information Systems Research, M.I.T.
6. Thomas A. Standish  
'Data Structure Techniques'  
Addison-Wesley Computer Science Series
7. Aho Ullman  
'Principles of Compiler Design'  
Addison-Wesley Computer Science Series
8. Tak To  
'SHELL: A Simulation for the Software Test Vehicle  
of the INFOPLEX Data Base Computer'  
Center for Information Systems Research, M.I.T.
9. Chat-Yu Lam, Stuart E. Madnick  
'INFOPLEX Data Base Computer Architecture'  
Center for Information Systems Research, M.I.T.
10. Bruce Blumberg  
'INFOSAM - A Sample Database Management System'  
Center for Information Systems Research, M.I.T.

APPENDIX

```

USINT: PROC OPTIONS (MAIN);
XINCLUDE DICTION;
XINCLUDE TOKEN;
DCL (NEWBUF, ACTCRD) ENTRY EXTERNAL;
DCL DCTNRY ENTRY (.CHAR (200) VAR)
    EXTERNAL RETURNS (CHAR (160) VAR);
DCL (LINE GARBAGE) CHAR (80);
DCL FIRSTLAST BIT(1) INIT ('0'B);
DCL (EXECBUFF,TRNSBUFF) CHAR (4000) VAR INIT ('');
DCL PLIST1(3) CHAR (8) INIT ('DVHJTL','TEST','HIGH(8)'),
    PLIST2(3) CHAR (8) INIT ('DVHJTL','CLEAR','HIGH(8)'),
    (HIGH,PLIRETC) BUILTIN;
RETCODE FIXED BINARY (31,0);
CMSCMD EXTERNAL ENTRY OPTIONS (ASSEMBLER INTER);
DCL JUNK CHAR (160) VAR;

JUNK = DCTNRY (DICTION,'INITIALIZE');
CALL CMSCMD (PLIST2.RETCODE);
PUT SKIP EDIT ('...INFOLEX DATA BASE MACHIN...')
    (COL(15),A);
PUT SKIP EDIT ('TYPE "VIR" FOR VIRTUAL INFORMATION PROCESSOR')
    (COL(1),A);
PUT SKIP EDIT ('TYPE "REAL" FOR REAL INFORMATION PROCESSOR')
    (COL(1),A);
GET EDIT (LINE) (COL(1),A(80));
LINE = TRANSLATE (LINE, 'ABCDEFGHIJKLMNORSTUVWXYZ',
    'abcdefghijklmnopqrstuvwxyz');
IF INDEX (LINE,'REAL') ^= 0 THEN RETURN;
ELSE IF INDEX (LINE,'VIR') = 0 THEN DO;
    PUT EDIT ('UNRECOGNIZABLE SUBSPACE NAME, PROGRAM TERMINATED!')
        (COL(1),A(80));
    RETURN;
END;

FIRSTLAST = '1'B;
DO WHILE ('1'B);
    CALL NEWBUF (EXECBUFF,TRNSBUFF,FIRSTLAST);
    IF ^FIRSTLAST THEN RETURN;
    CALL XBUFF (EXECBUFF);
END;

```

```

US100010
US100020
US100030
US100040
US100050
US100060
US100070
US100080
US100090
US100100
US100110
US100120
US100130
US100140
US100150
US100160
US100170
US100180
US100190
US100200
US100210
US100220
US100230
US100240
US100250
US100260
US100270
US100280
US100290
US100300
US100310
US100320
US100330
US100340
US100350
US100360
US100370
US100380
US100390
US100400
US100410
US100420
US100430
US100440
US100450
US100460

```



```

XBUFF: PROC (EXECBUFF);
DCL EXECBUFF CHAR (4000) VAR;
DCL (UNIT,RETSTMT) CHAR (2000) VAR;
DCL TKLSPTR PTR;
JUNK = DCINRY (DICTION,'ADHCNTRCLR');
TKLSPTR = NULL ();
DO WHILE (INDEX (EXECBUFF, ':') ~= 0);
    TKLSPTR = NULL();
    UNIT = GETS (EXECBUFF, ':') !! ':' ;
    IF INDEX (UNIT, 'RETRIEVE') ~= 0 THEN DO;
        CALL CMSCMD (PLIST2,RETSTMT);
        RETSTMT = UNIT ;
        CALL RETDSPY (RETSTMT) ;
    END;
    CALL ACTCRD (UNIT,TKLSPTR,DICTION);
END;
END XBUFF;

```

```

US100470
US100480
US100490
US100500
US100510
US100520
US100530
US100540
US100550
US100560
US100570
US100580
US100590
US100600
US100610
US100620
US100630
US100640
US100650
US100660
US100670
US100680
US100690

```

```

GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (2000) VAR):
DCL LIST CHAR (*) VAR,
TERM_ITEM CHAR (1),
RTN_LIST CHAR (2000) VAR,
I FIXED;

I = INDEX (LIST,TERM_ITEM);
IF I = 0 THEN DO;
RTN_LIST = LIST;
LIST = ',';
END;
ELSE DO;
RTN_LIST = SUBSTR (LIST,I,I-1);
LIST = SUBSTR (LIST,I+1);
END;
RETURN (RTN_LIST);
END GETS;

```

```

US100700
US100710
US100720
US100730
US100740
US100750
US100760
US100770
US100780
US100790
US100800
US100810
US100820
US100830
US100840
US100850
US100860
US100870
US100880
US100890
US100900
US100910
US100920
US100930

```

```

PRINTT: PROC (HEAD);
/* DEBUG TOOL TO PRINT OUT TOKEN CHAIN */
DCL (HEAD,M) PTR;

M = HEAD;
DO WHILE (M ^= NULL ());
  PUT SKIP EDIT ('TEST_TOKEN =$',M -> TOKEN.ITEM,'$$',
    'TEST_CLASS =$',M -> TOKEN.CLASS,'$$')
    (A,A,A,SKIP,A,A,A);
  M = M -> TOKEN.NEXT;
END;

END PRINTT;

```

```

US100940
US100950
US100960
US100970
US100980
US100990
US101000
US101010
US101020
US101030
US101040
US101050
US101060
US101070
US101080
US101090
US101100
US101110
US101120
US101130

```

```

RETDSPLY: PROC (DFNTN);
DCL DFNTN CHAR (2000) VAR;
DCL LINE CHAR (2000) VAR;
DCL I FIXED;
DCL SPACES FIXED INIT (0);
DCL SPACECH CHAR (2) VAR;
DO WHILE (DFNTN ^= '');
  LINE = GETS (DFNTN, ' ');
  IF LINE ^= '' THEN DO;
    CALL REPLACE (LINE, 'X1', ' ');
    CALL REPLACE (LINE, 'X2', ' ');
    CALL REPLACE (LINE, 'X3', ' ');
    CALL REPLACE (LINE, 'X5', ' ');
    CALL REPLACE (LINE, 'X4', ' ');
    IF SPACES ^= 0 THEN
      DO I = 1 TO SPACES;
        LINE = ' ' || LINE;
      END;
    PUT EDIT (LINE) (COL(1), A(80));
    IF DFNTN ^= '' THEN
      DFNTN = ' ' || DFNTN;
    END;
  ELSE DO;
    SPACECH = GETS (DFNTN, ' ');
    SPACES = FIXED (SPACECH);
  END;
END;
END RETDSPLY;

```

```

USIO1140
USIO1150
USIO1160
USIO1170
USIO1180
USIO1190
USIO1200
USIO1210
USIO1220
USIO1230
USIO1240
USIO1250
USIO1260
USIO1270
USIO1280
USIO1290
USIO1300
USIO1310
USIO1320
USIO1330
USIO1340
USIO1350
USIO1360
USIO1370
USIO1380
USIO1390
USIO1400
USIO1410
USIO1420
USIO1430
USIO1440
USIO1450
USIO1460
USIO1470

```

USIO1480  
 USIO1490  
 USIO1500  
 USIO1510  
 USIO1520  
 USIO1530  
 USIO1540  
 USIO1550  
 USIO1560  
 USIO1570  
 USIO1580  
 USIO1590  
 USIO1600  
 USIO1610  
 USIO1620  
 USIO1630  
 USIO1640  
 USIO1650  
 USIO1660  
 USIO1670  
 USIO1680  
 USIO1690  
 USIO1700  
 USIO1710  
 USIO1720  
 USIO1730  
 USIO1740  
 USIO1750  
 USIO1760  
 USIO1770

```

REPLACE: PROC (RLINE,FR_SYM,TO_SYM);
DCL (RLINE,REP_LINE) CHAR (2000) VAR;
FR_SYM CHAR (2) VAR;
TO_SYM CHAR (2) VAR;
I FIXED;

REP_LINE = '';
I = INDEX (RLINE,FR_SYM);
DO WHILE (I <= 0);
  IF LENGTH (FR_SYM) = 1 THEN DO;
    REP_LINE = REP_LINE || SUBSTR (RLINE,I - 1) || TO_SYM;
    RLINE = SUBSTR (RLINE,I + 1);
    I = INDEX (RLINE,FR_SYM);
  END;
ELSE DO;
  REP_LINE = REP_LINE || SUBSTR (RLINE,I - 1) || TO_SYM;
  RLINE = SUBSTR (RLINE,I + 2);
  I = INDEX (RLINE,FR_SYM);
END;
END;

RLINE = REP_LINE || RLINE;

END REPLACE;

END USINT;
  
```

```

NEW00010
NEW00020
NEW00030
NEW00040
NEW00050
NEW00060
NEW00070
NEW00080
NEW00090
NEW00100
NEW00110
NEW00120
NEW00130
NEW00140
NEW00150
NEW00160
NEW00170
NEW00180
NEW00190
NEW00200
NEW00210
NEW00220
NEW00230
NEW00240
NEW00250
NEW00260
NEW00270
NEW00280
NEW00290
NEW00300
NEW00310
NEW00320
NEW00330
NEW00340
NEW00350
NEW00360
NEW00370
NEW00380
NEW00390
NEW00400
NEW00410
NEW00420
NEW00430
NEW00440
NEW00450
NEW00460
NEW00470
NEW00480
NEW00490
NEW00500
NEW00510
NEW00520
NEW00530
NEW00540
NEW00550
NEW00560
NEW00570
NEW00580

```

```

NEWBUF: PROC (EXECBUFF,TRNSBUFF,FIRSTLAST);
/******
/* THIS PROGRAM INTERACTS WITH THE USER WITHIN THE VIRTUAL
/* INFORMATION SUBSYSTEM: IT IS CALLED BY THE GENERAL USER
/* INTERFACE PROGRAM WHICH ASKS THE USER WHETHER HE WISHES TO
/* ENTER THE VIRTUAL INFO SUBSYSTEM OR THE REAL INFO SUBSYSTEM */
/******
/* FORMAL PARAMETERS EXECBUFF AND TRNSBUFF REPRESENT
/* THE EXECUTION AND TRANSACTION BUFFERS. ON A "RUNTRANS" OR
/* "DDEL" COMMAND THE TRANSACTION BUFFER WILL BE PUT INTO THE
/* EXECUTION BUFFER, AND THE ROUTINE RETURNS. OTHERWISE
/* THE ROUTINE WILL RETURN AS SOON AS THERE IS AN EXECUTABLE
/* STATEMENT WITHIN THE EXECUTION BUFFER.
/* FORMAL PARAMETER "FIRSTLAST" REPRESENTS THE BIT WHICH
/* WILL BE TURNED ON WHEN THE "TERMINATE" COMMAND IS ISSUED
/* AND WHICH WILL BE CHECKED BY THE "USER" PROGRAM TO
/* TERMINATE SUBSEQUENT CALLS TO THE "BUFFER" PROGRAM.
/* BUFFER COMMANDS SUCH AS "FINPUT", "FSAVE", "TRANSACTION"
/* "END", ..... WILL BE HANDLED ENTIRELY WITHIN THIS
/* ROUTINE: THUS, THESE COMMANDS WILL NOT BE INCLUDED
/* WITHIN THE "BUFFER" PARAMETER WHICH IS PASSED BACK TO THE
/* "USINT" PROGRAM
/* THIS BUFFER PROGRAM FEATURES AN INTERACTIVE LINE EDITOR
/* THROUGH WHICH A USER ACCESSES THE TRANSACTION BUFFER
/******
DCL PLIST1(3) CHAR(8) INIT ('DVHUTL', 'TEST', HIGH(8)),
  PLIST2(3) CHAR(8) INIT ('DVHUTL', 'CLEAR', HIGH(8)),
  (HIGH, PLIRETC) BUILTIN,
  RETCODE FIXED BINARY (31,0), /* TO CLEAR SCREEN */
  CMSCMD EXTERNAL ENTRY OPTIONS (ASSEMBLER INTER);
DCL (EXECBUFF,TRNSBUFF) CHAR (4000) VAR;
DCL (FIRSTLAST,STATUS) BIT (1);
DCL (WORD,GARBAGE,KEY) CHAR (80) VAR;
DCL PRSTLINE CHAR (80);
DCL (CPLNVAR,STRNSP) CHAR (80) VAR;
DCL (BLANKLINE,ERROR) BIT(1) INIT ('0'B);
DCL (I,I1,LDSPPACES) FIXED;
DCL TRFILE FILE RECORD
  ENV (FB RECSIZE (80) BLKSIZE (800));
DCL L_DF_CMDS (13) CHAR (12) VAR;
DCL LIST_OF_CMDS (13) CHAR (12) VAR
  INIT ('RUNTRANS',
        'FSAVE',
        'TERMINATE',
        'TRANSACTION',
        'ENDTRANS',
        'FINPUT',
        'KILLEXEC',
        'ERASETRANS',
        'DDELTRANS',
        'HELP');

```

NEW00590  
 NEW00600  
 NEW00610  
 NEW00620  
 NEW00630  
 NEW00640  
 NEW00650  
 NEW00660  
 NEW00670  
 NEW00680  
 NEW00690  
 NEW00700  
 NEW00710  
 NEW00720  
 NEW00730  
 NEW00740  
 NEW00750  
 NEW00760  
 NEW00770  
 NEW00780  
 NEW00790  
 NEW00800  
 NEW00810  
 NEW00820  
 NEW00830  
 NEW00840  
 NEW00850  
 NEW00860  
 NEW00870  
 NEW00880  
 NEW00890  
 NEW00900  
 NEW00910  
 NEW00920  
 NEW00930  
 NEW00940  
 NEW00950  
 NEW00960  
 NEW00970  
 NEW00980  
 NEW00990  
 NEW01000  
 NEW01010  
 NEW01020  
 NEW01030  
 NEW01040  
 NEW01050  
 NEW01060  
 NEW01070  
 NEW01080  
 NEW01090  
 NEW01100  
 NEW01110  
 NEW01120  
 NEW01130  
 NEW01140  
 NEW01150  
 NEW01160

```

    'INSERT',
    'DELETE',
    'TOPLINE');
DCL EXECB BIT (1) INIT ('1'B);
DCL (LINENM,TOPLINE) FIXED;
DCL 1 EXECBMS,
    2 LIMIT FIXED,
    2 LOC (0:100) FIXED;
DCL 1 TRANSMKS LIKE EXECBMS;

EXECB = '1'B;
L_OF_CMDS (*) = LIST_OF_CMDS (*);
CALL SETMKS (TRNSBUFF,TRANSMKS);
CALL SETMKS (EXECBUFF,EXECBMS);
TOPLINE = 0;
DO WHILE ('1'B);
  ERROR = '0'B;
  CALL CMSCMD (PLIST2,RETCODE); /* CLEAR CREEN */
  PUT SKIP EDIT
    ('***TRANSACTION BUFFER***')
    (COL (25),A);
  IF EXECB THEN DO;
    CALL BDISPLAY (TRNSBUFF,TOPLINE,5,TRANSMKS,'1'B);
    PUT SKIP EDIT
      ('***EXECUTION BUFFER***')
      (COL (25),A);
    CALL BDISPLAY (EXECBUFF,0,12,EXECBMS,'0'B);
  END;
ELSE
  CALL BDISPLAY (TRNSBUFF,TOPLINE,18,TRANSMKS,'1'B);
  GET EDIT (PRSTLINE) (COL(1),A(80));
  CPLNVAR = TRANSLATE (PRSTLINE, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
    'abcdefghijklmnopqrstuvwxyz');
  CALL REPLACE (CPLNVAR,'X','X4');
  CALL REPLACE (CPLNVAR,' ','X0');
  CALL REPLACE (CPLNVAR,' ','X3');
  CALL REPLACE (CPLNVAR,'@','X5');

  LDSPACES = 0;
  BLANKLINE = '0'B;
  CALL KBLKS;
  KEY = NEXTWORD;
  IF ERROR THEN GOTO FNDLOOP;
  IF LENGTH (KEY) >= 1 THEN
    L_OF_CMDS (*) = SUBSTR (LIST_OF_CMDS (*),
      1,
      MIN (LENGTH (KEY),
        LENGTH (LIST_OF_CMDS (*))));

  SELECT (KEY);
  WHEN (L_OF_CMDS (1)) DO;
    EXECBUFF = TRNSBUFF;
  RETURN;
END;
  
```

```

      WHEN (L_OF_CMDS (2))
      CALL FSARE;
      WHEN (L_OF_CMDS (3)) DO:
      FIRSTLAST = 'O'B;
      RETURN;
      END;
      WHEN (L_OF_CMDS (4)) DO:
      STRNSP = '';
      EXECB = 'O'B;
      END;
      WHEN (L_OF_CMDS (5)) DO:
      EXECB = 'I'B;
      STRNSP = '';
      END;
      WHEN (L_OF_CMDS (6))
      CALL_FINPUT;
      WHEN (L_OF_CMDS (7)) DO:
      EXECBUFF = '';
      CALL SETMKS (EXECBUFF,EXECMKS);
      END;
      WHEN (L_OF_CMDS (8)) DO:
      TRANSBUFF = '';
      CALL SETMKS (TRANSBUFF,TRANSMKS);
      END;
      WHEN (L_OF_CMDS (9)) DO:
      EXECBUFF = TRANSBUFF;
      TRANSBUFF = '';
      RETURN;
      END;
      WHEN (L_OF_CMDS (10))
      CALL_HELP;
      WHEN (L_OF_CMDS (11))
      IF WITHNLM (NEXTWORD,TRANSMKS.LIMIT,LINENM) THEN DO:
      CALL KBLKS;
      STATUS = BUILDBUF (TRANSBUFF,TRANSMKS,LINENM);
      END;
      ELSE
      /* MSG */;
      WHEN (L_OF_CMDS (12))
      IF WITHNLM (NEXTWORD,TRANSMKS.LIMIT - 1,LINENM) THEN
      CALL DELTE;
      ELSE /* MSG */;
      WHEN (L_OF_CMDS (13))
      IF WITHNLM (NEXTWORD,TRANSMKS.LIMIT,LINENM) THEN
      TOPLINE = LINENM;
      ELSE /* MSG */;
      OTHERWISE DO:
      CPLNVAR = STRNSP;
      IF EXECB THEN
      IF BUILDBUF (EXECBUFF,EXECMKS,EXECMKS.LIMIT) THEN
      RETURN;

```

```

NEW01170
NEW01180
NEW01190
NEW01200
NEW01210
NEW01220
NEW01230
NEW01240
NEW01250
NEW01260
NEW01270
NEW01280
NEW01290
NEW01300
NEW01310
NEW01320
NEW01330
NEW01340
NEW01350
NEW01360
NEW01370
NEW01380
NEW01390
NEW01400
NEW01410
NEW01420
NEW01430
NEW01440
NEW01450
NEW01460
NEW01470
NEW01480
NEW01490
NEW01500
NEW01510
NEW01520
NEW01530
NEW01540
NEW01550
NEW01560
NEW01570
NEW01580
NEW01590
NEW01600
NEW01610
NEW01620
NEW01630
NEW01640
NEW01650
NEW01660
NEW01670
NEW01680
NEW01690
NEW01700
NEW01710
NEW01720

```



NEW01730  
NEW01740  
NEW01750  
NEW01760  
NEW01770  
NEW01780  
NEW01790  
NEW01800  
NEW01810  
NEW01820  
NEW01830  
NEW01840

ELSE;  
ELSE  
STATUS = BUILD0UF (TRNSBUUF, TRANSMKS, TRANSMKS.LIMIT);  
END;  
END;  
ENDLOOP;  
END;

NEWO1850  
 NEWO1860  
 NEWO1870  
 NEWO1880  
 NEWO1890  
 NEWO1900  
 NEWO1910  
 NEWO1920  
 NEWO1930  
 NEWO1940  
 NEWO1950  
 NEWO1960  
 NEWO1970  
 NEWO1980  
 NEWO1990  
 NEWO2000  
 NEWO2010  
 NEWO2020  
 NEWO2030  
 NEWO2040  
 NEWO2050  
 NEWO2060

```

LDSPCH: PROC (LDSPACES) RETURNS (CHAR (20) VAR):
/* THIS ROUTINE CONSTRUCTS THE LINE HEADER FOR EACH */
/* BUFFER LINE, CHAR " " || NUMBER OF LEADING SPACES || CHAR ":" */
DCL (LDSPACES.I) FIXED;
DCL LDSPCHAR CHAR (20) VAR;
LDSPCHAR = CHAR (LDSPACES);
DO I = 1 TO LENGTH (LDSPCHAR);
  IF SUBSTR (LDSPCHAR,I,1) ^= ' ', THEN DO;
    LDSPCHAR = SUBSTR (LDSPCHAR,I);
    I = 100;
  END;
END;
LDSPCHAR = ' ' || LDSPCHAR || ':' ;
RETURN (LDSPCHAR);
END LDSPCH;
  
```

NEW02070  
 NEW02080  
 NEW02090  
 NEW02100  
 NEW02110  
 NEW02120  
 NEW02130  
 NEW02140  
 NEW02150  
 NEW02160  
 NEW02170  
 NEW02180  
 NEW02190  
 NEW02200  
 NEW02210  
 NEW02220  
 NEW02230  
 NEW02240  
 NEW02250  
 NEW02260  
 NEW02270  
 NEW02280  
 NEW02290  
 NEW02300  
 NEW02310  
 NEW02320  
 NEW02330  
 NEW02340  
 NEW02350

```

BUILDUF: PROC (BUFF,MKS,LNUM) RETURNS (BIT (1));
/* THIS ROUTINE BUILDS UP EITHER THE EXECUTION OR THE */
/* TRANSACTION BUFFER ONE LINE AT A TIME */
DCL BUFF CHAR (4000) VAR;
DCL 1 MKS;
      2 LIMIT FIXED;
      2 LOC (0:100) FIXED;
DCL (LNUM,LNUM1,LEN,I) FIXED;
CPLNVAR = LDSPCH (LDSPACES) || STRNSP || ' ';
CALL TRNSLATE;
LEN = LENGTH (CPLNVAR);
IF LEN = 0 THEN
  RETURN ('O'B);
LNUM1 = LNUM;
MKS.LIMIT = MKS.LIMIT + 1;
DO I = MKS.LIMIT TO LNUM1 + 1 BY -1;
  MKS.LOC (I) = MKS.LOC (I-1) + LEN;
END;
BUFF = SUBSTR (BUFF,I,MKS.LOC (LNUM1) - 1)
      || CPLNVAR
      || SUBSTR (BUFF,MKS.LOC (LNUM1));
RETURN (INDEX (CPLNVAR,':') ^= 0);
END BUILDUF;
  
```

```

NEW02360
NEW02370
NEW02380
NEW02390
NEW02400
NEW02410
NEW02420
NEW02430
NEW02440
NEW02450
NEW02460
NEW02470
NEW02480
NEW02490
NEW02500
NEW02510
NEW02520
NEW02530
NEW02540
NEW02550
NEW02560
NEW02570
NEW02580
NEW02590
NEW02600
NEW02610
NEW02620
NEW02630
NEW02640
NEW02650
NEW02660
NEW02670
NEW02680
NEW02690
NEW02700
NEW02710
NEW02720
NEW02730
NEW02740
NEW02750
NEW02760
NEW02770
NEW02780
NEW02790
NEW02800
NEW02810
NEW02820
NEW02830
NEW02840
NEW02850
NEW02860
NEW02870
NEW02880
NEW02890
NEW02900
NEW02910
NEW02920
NEW02930
NEW02940

TRANSLATE PROC; /* STATUS BEING 'O'B UPON ENTRY
/* THIS PROGRAM CHANGES QUOTE AND SEMICOLON CHARACTERS WITHIN
/* STRING CONSTANTS TO SPECIAL CHARACTERS, AND ALSO THE SEMICOLON
/* CHARACTERS WITHIN "COMMENT STATEMENTS TO A SPECIAL SYMBOL
/* ROUTINE ACTS ON GLOBAL VARIABLE "CPLNVAR"
DCL (TEMP1,COPY) CHAR (80) VAR INIT ('');
DCL (QUOTE, QUOTE1, SLSH, SLSH1, SCOLON, J) FIXED (4);
DO WHILE ('1'B);
    SLSH = INDEX (CPLNVAR, '\');
    QUOTE = INDEX (CPLNVAR, ' ');
    SCOLON = INDEX (CPLNVAR, ';');
    IF SLSH=0 & QUOTE=0 THEN GOTO DONE;
    IF QUOTE=0 & (SLSH=0 | QUOTE<SLSH) THEN
        DO;
            COPY = COPY || SUBSTR (CPLNVAR,1,QUOTE);
            TEMP1 = SUBSTR (CPLNVAR,QUOTE+1);
            DO WHILE (INDEX (TEMP1, ' ') = 0);
                IF INDEX (TEMP1, ' ') = 1
                    THEN TEMP1 = 'X1' || SUBSTR (TEMP1,3);
                ELSE TEMP1 = SUBSTR (TEMP1,1,INDEX (TEMP1, ' ') - 1)
                    || 'X1' ||
                    SUBSTR (TEMP1, INDEX (TEMP1, ' ') + 2);
            END;
            QUOTE1 = INDEX (TEMP1, ' ');
            IF QUOTE1 = 0 THEN
                DO;
                    CPLNVAR = '';
                    COPY = '';
                    CALL CMSCMD (PLIST2.RETCODE);
                    PUT SKIP EDIT ('MISSING QUOTE TERMINATOR') (A);
                    PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);
                    GET EDIT (GARBAGE) (A(80));
                    GOTO DONE;
                END;
            TEMP1 = SUBSTR (TEMP1,1,QUOTE1);
            DO J = 1 TO QUOTE1;
                IF SUBSTR (TEMP1,J,1) = ' ';
                    THEN COPY = COPY || 'X2' ;
                ELSE COPY = COPY || SUBSTR (TEMP1,J,1);
            END;
            CPLNVAR = SUBSTR (CPLNVAR,QUOTE+QUOTE1+1);
        END;
    IF SLSH=0 & (QUOTE=0 | SLSH<QUOTE) THEN
        DO;
            COPY = COPY || SUBSTR (CPLNVAR,1,SLSH);
            TEMP1 = SUBSTR (CPLNVAR,SLSH+1);
            SLSH1 = INDEX (TEMP1, '\');
            IF SLSH1 = 0 THEN DO;
                COPY = '';
                CPLNVAR = '';
                CALL CMSCMD (PLIST2.RETCODE);
                PUT SKIP EDIT ('ERROR* MISSING COMMENT TERMINATOR') (A);
                PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);
            END;
        END;

```

NEW02950  
 NEW02960  
 NEW02970  
 NEW02980  
 NEW02990  
 NEW03000  
 NEW03010  
 NEW03020  
 NEW03030  
 NEW03040  
 NEW03050  
 NEW03060  
 NEW03070  
 NEW03080  
 NEW03090  
 NEW03100  
 NEW03110  
 NEW03120  
 NEW03130  
 NEW03140  
 NEW03150  
 NEW03160  
 NEW03170  
 NEW03180  
 NEW03190

```

      GET EDIT (GARBAGE) (A(80));
      RETURN;
    END;
    TEMP1 = SUBSTR (TEMP1.1.SLSH1);
    DO J = 1 TO SLSH1;
      IF SUBSTR (TEMP1.J.1) = ',';
        THEN COPY = COPY || 'X2';
        ELSE COPY = COPY || SUBSTR (TEMP1.J.1);
      END;
      CPLNVAR = SUBSTR (CPLNVAR, SLSH+SLSH1+1);
    END;
  END;

  DONE:
    CPLNVAR = COPY || CPLNVAR ;
  END TRANSLATE;

```

```

FINPUT: PROC;
  DCL PRSTLINE CHAR (80);
  DCL TRFILE FILE RECORD
    ENV (FB RECSIZE (80) BLKSIZE (800));
  DCL FNAME CHAR (80) VAR;
  ON ENDFILE (TRFILE) GOTO FINEND;
  ON TRANSMIT (TRFILE) GOTO ERRPT;
  CALL CMSCMD (PLIST2, RETCODE);
  FNAME = NEXTWORD;
  FNAME = TRANSLATE (FNAME,
    .....
    '.....';
  IF INDEX (FNAME, '..') = 0
    | ERROR
    | LENGTH (FNAME) > 8 THEN
    ERRPT: DO;
      PUT SKIP EDIT
        ('INVALID FILE TYPE! FILE NOT ENTERED INTO BUFFER') (A);
      PUT SKIP EDIT
        ('TYPE "ENTER" KEY TO CONTINUE') (A);
      GET EDIT (GARBAGE) (COL(1).A(80));
      RETURN;
    END;
  PUT SKIP EDIT ('READING TRANSACTION BUFFER FROM CMS FILE:') (A);
  PUT SKIP EDIT ('FILE ' || FNAME || ' ') (COL(30).A);
  PUT SKIP EDIT ('OLD TRANSACTION BUFFER CONTENT DELETED') (A);
  CALL WAIT;
  OPEN FILE (TRFILE) TITLE (FNAME) INPUT;
  TRNSBUFF = '';
  DO WHILE ('1'B);
    READ FILE (TRFILE) INTO (PRSTLINE);
    CPLNVAR = TRANSLATE (PRSTLINE, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ');
    CALL REPLACE (CPLNVAR, 'X', 'X4');
    CALL REPLACE (CPLNVAR, 'Y', 'Y0');
    CALL REPLACE (CPLNVAR, 'Z', 'Z3');
    CALL REPLACE (CPLNVAR, '0', 'X5');
    LDSPACES = 0;
    BLANKLINE = '0'B;
    CALL KBLKS;
    CPLNVAR = LDSPCH (LDSPACES) || STNSP;
    CALL TRANSLATE;
    TRNSBUFF = TRNSBUFF || CPLNVAR;
  END;
  FINEND;
  CALL SETMKS (TRNSBUFF, TRANSMKS);
  CLOSE FILE (TRFILE);
END FINPUT;

```

NEW03200  
 NEW03210  
 NEW03220  
 NEW03230  
 NEW03240  
 NEW03250  
 NEW03260  
 NEW03270  
 NEW03280  
 NEW03290  
 NEW03300  
 NEW03310  
 NEW03320  
 NEW03330  
 NEW03340  
 NEW03350  
 NEW03360  
 NEW03370  
 NEW03380  
 NEW03390  
 NEW03400  
 NEW03410  
 NEW03420  
 NEW03430  
 NEW03440  
 NEW03450  
 NEW03460  
 NEW03470  
 NEW03480  
 NEW03490  
 NEW03500  
 NEW03510  
 NEW03520  
 NEW03530  
 NEW03540  
 NEW03550  
 NEW03560  
 NEW03570  
 NEW03580  
 NEW03590  
 NEW03600  
 NEW03610  
 NEW03620  
 NEW03630  
 NEW03640  
 NEW03650  
 NEW03660  
 NEW03670  
 NEW03680  
 NEW03690  
 NEW03700  
 NEW03710  
 NEW03720  
 NEW03730

```

KBLKS: PROC;
/* SETS UP "STRNSP" FROM "CPLNVAR" */
/* CALCULATES "LDSPACES" */
DCL I FIXED;
DO I = 1 TO LENGTH (CPLNVAR);
  IF SUBSTR (CPLNVAR,I,1) ^= ' '
  THEN DO;
    LDSPACES = I - 1;
    I = 150;
  END;
  ELSE IF I = LENGTH (CPLNVAR) THEN BLANKLINE = '1'B;
END;
IF BLANKLINE = '0'B
THEN DO;
  CPLNVAR = SUBSTR (CPLNVAR, LDSPACES + 1);
  DO I = 1 TO LENGTH (CPLNVAR) TO 1 BY -1;
    IF SUBSTR (CPLNVAR,I,1) ^= ' '
    THEN DO;
      CPLNVAR = SUBSTR (CPLNVAR,I,I);
      I = 0;
    END;
  END;
  STRNSP = CPLNVAR;
END;
ELSE DO;
  STRNSP = '';
  LDSPACES = 0;
END;
END KBLKS;

```

```

NEW03740
NEW03750
NEW03760
NEW03770
NEW03780
NEW03790
NEW03800
NEW03810
NEW03820
NEW03830
NEW03840
NEW03850
NEW03860
NEW03870
NEW03880
NEW03890
NEW03900
NEW03910
NEW03920
NEW03930
NEW03940
NEW03950
NEW03960
NEW03970
NEW03980
NEW03990
NEW04000
NEW04010
NEW04020
NEW04030
NEW04040
NEW04050
NEW04060
NEW04070
NEW04080

```

```

GETS: PROC (LIST, TERM_ITEM) RETURNS (CHAR (80) VAR):
DCL LIST CHAR (*) VAR;
TERM_ITEM CHAR (1);
RTN_LIST CHAR (80) VAR;
I FIXED;

I = INDEX (LIST, TERM_ITEM);
IF I = 0 THEN DO;
RTN_LIST = LIST;
LIST = '';
END;
ELSE DO;
RTN_LIST = SUBSTR (LIST, I, I - 1);
LIST = SUBSTR (LIST, I + 1);
END;
RETURN (RTN_LIST);
END GETS;

```

```

NEW04090
NEW04100
NEW04110
NEW04120
NEW04130
NEW04140
NEW04150
NEW04160
NEW04170
NEW04180
NEW04190
NEW04200
NEW04210
NEW04220
NEW04230
NEW04240
NEW04250
NEW04260
NEW04270
NEW04280
NEW04290
NEW04300

```



```

NEW04310
NEW04320
NEW04330
NEW04340
NEW04350
NEW04360
NEW04370
NEW04380
NEW04390
NEW04400
NEW04410
NEW04420
NEW04430
NEW04440
NEW04450
NEW04460
NEW04470
NEW04480
NEW04490
NEW04500
NEW04510
NEW04520
NEW04530
NEW04540
NEW04550
NEW04560
NEW04570
NEW04580
NEW04590
NEW04600
NEW04610
NEW04620
NEW04630
NEW04640
NEW04650
NEW04660
NEW04670
NEW04680
NEW04690
NEW04700

```

```

NEXTWORD: PROC RETURNS (CHAR (80) VAR);
/* THIS ROUTINE RETURNS A NULL STRING IF CPLNVAR IS NULL,
/* RETURNS THE STRING '' IF CPLNVAR IS A BLANKLINE,
/* OTHERWISE, IT RETURNS THE FIRST WORD OF CPLNVAR
/* DELIMITED BY AN OPTIONAL SUCCEEDING BLANKLINE CHARACTER
*/
DCL NWORD CHAR (80) VAR;
DCL SYMBOL CHAR (1);
IF CPLNVAR = '' THEN RETURN ('');
DO WHILE ('1'B);
CALL RMVFLKS;
IF CPLNVAR = '' THEN RETURN ('');
IF INDEX ('\@', SUBSTR (CPLNVAR, 1, 1)) ^= 0 THEN
DO WHILE (INDEX ('\@', SUBSTR (CPLNVAR, 1, 1)) ^= 0);
SYMBOL = SUBSTR (CPLNVAR, 1, 1);
CPLNVAR = SUBSTR (CPLNVAR, 2);
IF SYMBOL = '\.' THEN
IF INDEX (CPLNVAR, '\.') = 0 THEN DO;
ERROR = '1';
CPLNVAR = '';
PUT SKIP EDIT ('MISSING COMMENT TERMINATOR') (A);
PUT SKIP EDIT ('TYPE "ENTER" TO CONTINUE') (A);
GET EDIT (GARBAGE) (A(80));
RETURN;
END;
ELSE
GARBAGE = GETS (CPLNVAR, SYMBOL);
ELSE
GARBAGE = GETS (CPLNVAR, '.');
END;
IF SUBSTR (CPLNVAR, 1, 1) ^= ' ' THEN DO;
NWORD = GETS (CPLNVAR, ' ');
RETURN (NWORD);
END;
END;

```

NEW04710  
 NEW04720  
 NEW04730  
 NEW04740  
 NEW04750  
 NEW04760  
 NEW04770  
 NEW04780  
 NEW04790  
 NEW04800  
 NEW04810  
 NEW04820  
 NEW04830  
 NEW04840  
 NEW04850  
 NEW04860  
 NEW04870  
 NEW04880

```

RMVFBKLS: PROC;
DCL I FIXED;
IF CPLNVAR = '' THEN RETURN;
DO I = 1 TO LENGTH (CPLNVAR);
  IF SUBSTR (CPLNVAR,I,1) ^= ' ' THEN DO;
    CPLNVAR = SUBSTR (CPLNVAR, I);
    RETURN;
  END;
END;
CPLNVAR = '';
RETURN;
END RMVFBKLS;
END NEXTWORD;
  
```

```

NEW04890
NEW04900
NEW04910
NEW04920
NEW04930
NEW04940
NEW04950
NEW04960
NEW04970
NEW04980
NEW04990
NEW05000
NEW05010
NEW05020
NEW05030
NEW05040
NEW05050
NEW05060
NEW05070
NEW05080
NEW05090
NEW05100
NEW05110
NEW05120
NEW05130
NEW05140
NEW05150
NEW05160
NEW05170
NEW05180
NEW05190
NEW05200
NEW05210
NEW05220
NEW05230
NEW05240
NEW05250
NEW05260
NEW05270
NEW05280
NEW05290
NEW05300
NEW05310
NEW05320
NEW05330
NEW05340
NEW05350
NEW05360
NEW05370
NEW05380
NEW05390
NEW05400
NEW05410
NEW05420
NEW05430
NEW05440
NEW05450
NEW05460
NEW05470

```

```

FSAVE: PROC;
DCL FNAME CHAR (80) VAR;
DCL TRFILE FILE RECORD
ENV (FB RECSIZE (80) BLKSIZE (800));
DCL LDSPCHAR CHAR (20) VAR;
TRANSBUF = TRANSBUF || LDSPCH (0);
ON ENDFILE (TRFILE) GOTO FSVEND;
CALL CMSCMD (PLIST2.RETCODE);
FNAME = NEXTWORD;
FNAME = TRANSLATE (FNAME,
'.....';
'~!@%$^&*()_+-=<~\|{}'";:/'?.,<>');
IF INDEX (FNAME, '~') ^= 0
| ERROR
| LENGTH (FNAME) > 8 THEN DO;
PUT SKIP EDIT
('INVALID FILE NAME! BUFFER NOT SAVED') (A);
DO SKIP EDIT
('TYPE "ENTER" KEY TO CONTINUE') (A);
GET EDIT (GARBAGE) (COL(1),A(80));
GOTO FSVEND;
END;
PUT SKIP EDIT ('SAVING TRANSACTION BUFFER INTO CMS FILE:') (A);
PUT SKIP EDIT ('FILE ' || FNAME || ' ') (COL(30),A);
CALL WAIT;
OPEN FILE (TRFILE) TITLE (FNAME) OUTPUT;
CPLNVAR = '';
DO WHILE (TRANSBUF ^= '');
TRANSBUF = SUBSTR (TRANSBUF,2);
LDSPCHAR = GETS (TRANSBUF,');
IF SUBSTR (TRANSBUF,1,1) = '0' THEN
IF LDSPCHAR = '0' THEN DO;
CPLNVAR = '';
PRSTLINE = CPLNVAR;
CPLNVAR = '';
WRITE FILE (TRFILE) FROM (PRSTLINE);
END;
ELSE DO;
DO I = 1 TO FIXED (LDSPCHAR);
CPLNVAR = CPLNVAR || ',';
END;
PRSTLINE = CPLNVAR;
CPLNVAR = '';
WRITE FILE (TRFILE) FROM (PRSTLINE);
END;
ELSE IF TRANSBUF ^= '' THEN DO;
IF LDSPCHAR ^= '0' THEN
DO I = 1 TO FIXED (LDSPCHAR);
CPLNVAR = CPLNVAR || ',';
END;
CPLNVAR = CPLNVAR || GETS (TRANSBUF,');
TRANSBUF = '0' || TRANSBUF;
CALL REPLACE (CPLNVAR, '%', ',');
CALL REPLACE (CPLNVAR, '%2', ',');

```

NEW05480  
 NEW05490  
 NEW05500  
 NEW05510  
 NEW05520  
 NEW05530  
 NEW05540  
 NEW05550  
 NEW05560  
 NEW05570  
 NEW05580  
 NEW05590  
 NEW05600  
 NEW05610  
 NEW05620  
 NEW05630  
 NEW05640  
 NEW05650  
 NEW05660  
 NEW05670

```

CALL REPLACE (CPLNVAR, 'X1', ' ');
CALL REPLACE (CPLNVAR, 'X3', ' ');
CALL REPLACE (CPLNVAR, 'X5', ' ');
CALL REPLACE (CPLNVAR, 'X4', ' ');
PRSTLINE = CPLNVAR;
CPLNVAR = '';
WRITE FILE (TRFILE) FROM (PRSTLINE);
END;

END;
FSVEND;
CLOSE FILE (TRFILE);
CALL SETWKS (TRNSBUFF, TRANSMKS);
END FSAVE;
  
```

NEW05680  
NEW05690  
NEW05700  
NEW05710  
NEW05720  
NEW05730  
NEW05740  
NEW05750  
NEW05760  
NEW05770  
NEW05780  
NEW05790  
NEW05800  
NEW05810  
NEW05820

```
WAIT: PROC:
/* THIS PROGRAM SIMPLY DELAYS EXECUTION BY A FEW SECONDS */
/* USED TO HOLD DISPLAY LONG ENOUGH TO BE RECOGNIZED */
DCL (I,I1) FIXED;
DO I = 1 TO 500;
    DO I1 = 1 TO 500;
        END;
    END;
END WAIT;
```

REPLACE: PROC (RLINE,FR\_SYM,TO\_SYM);

DCL (RLINE,REP\_LINE) CHAR (80) VAR,  
FR\_SYM CHAR (2) VAR,  
TO\_SYM CHAR (2) VAR,  
I FIXED;

REP\_LINE = '';

I = INDEX (RLINE,FR\_SYM);

DO WHILE (I <= 0);

IF LENGTH (FR\_SYM) = 1 THEN DO;

REP\_LINE = REP\_LINE || SUBSTR (RLINE,I - 1) || TO\_SYM;

RLINE = SUBSTR (RLINE,I + 1);

I = INDEX (RLINE,FR\_SYM);

END;

ELSE DO;

REP\_LINE = REP\_LINE || SUBSTR (RLINE,I - 1) || TO\_SYM;

RLINE = SUBSTR (RLINE,I + 2);

I = INDEX (RLINE,FR\_SYM);

END;

END;

RLINE = REP\_LINE || RLINE;

END REPLACE;

NEW05830  
NEW05840  
NEW05850  
NEW05860  
NEW05870  
NEW05880  
NEW05890  
NEW05900  
NEW05910  
NEW05920  
NEW05930  
NEW05940  
NEW05950  
NEW05960  
NEW05970  
NEW05980  
NEW05990  
NEW06000  
NEW06010  
NEW06020  
NEW06030  
NEW06040  
NEW06050  
NEW06060  
NEW06070  
NEW06080  
NEW06090  
NEW06100  
NEW06110

NEW06120  
NEW06130  
NEW06140  
NEW06150  
NEW06160  
NEW06170  
NEW06180  
NEW06190  
NEW06200  
NEW06210  
NEW06220  
NEW06230  
NEW06240  
NEW06250  
NEW06260  
NEW06270  
NEW06280  
NEW06290  
NEW06300  
NEW06310  
NEW06320  
NEW06330  
NEW06340  
NEW06350  
NEW06360  
NEW06370  
NEW06380  
NEW06390  
NEW06400  
NEW06410

```

HELP: PROC;
CALL CHSCMD (PLIST2,RETCODE);
PUT SKIP EDIT
  ("RUNTRANS" EXECUTE TRANSACTION BUFFER', BUFFER DELETED',
  "ODELTRANS" EXECUTE TRANSACTION BUFFER FROM FILE "FILE (ARG)"',
  "FINPUT"(ARG) SAVE TRANSACTION BUFFER INTO FILE "FILE (ARG)"',
  "FSAVE"(ARG) TERMINATE PROGRAM - RETURN TO CMS',
  "TERMINATE" ENTER CURRENT TRANSACTION BUFFER',
  "TRANSACTION" EXIT TRANSACTION BUFFER',
  "ENDTRANS" ERASE CURRENT TRANSACTION BUFFER',
  "ERASETRANS" KILL CURRENT EXECUTION BUFFER',
  "KILLEXEC" BRIEF EXPLANATION OF COMMANDS',
  "HELP" INSERT "STR" STRING INTO TRANSACTION BUFFER AT',
  "INSERT" (NUM)(STR) LINE NUMBER "NUM",
  "DELETE"(NUM) DELETE (NUM)TH LINE FROM TRANSACTION BUFFER',
  "TOPLINE"(NUM) START DISPLAY FROM (NUM)TH LINE OF BUFFER')
  (14 (SKIP.A));
PUT SKIP EDIT
  ('EACH COMMAND MAY BE IDENTIFIED BY TWO OR MORE OF ITS LEADING' !!
  ' CHARACTERS.')
  (A);
1 PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);
2 GET EDIT (GARBAGE) (A(80));
3 END HELP;

```

NEW06420  
 NEW06430  
 NEW06440  
 NEW06450  
 NEW06460  
 NEW06470  
 NEW06480  
 NEW06490  
 NEW06500  
 NEW06510  
 NEW06520  
 NEW06530  
 NEW06540  
 NEW06550  
 NEW06560  
 NEW06570  
 NEW06580  
 NEW06590  
 NEW06600  
 NEW06610  
 NEW06620  
 NEW06630  
 NEW06640  
 NEW06650

```

SETMKS: PROC (BUFF,MKS):
DCL BUFF CHAR (4000) VAR;
DCL (I,J,LASTMARK,NOWMARK) FIXED;
DCL I MKS,
      2 LIMIT FIXED,
      2 LOC (0:100) FIXED;
MKS.LOC (*) = 0;
J = 0;
LASTMARK = 0;
NOWMARK = INDEX (BUFF,'@');
DO WHILE (NOWMARK ^= 0);
  LASTMARK = LASTMARK + NOWMARK;
  MKS.LOC (J) = LASTMARK;
  J = J + 1;
  NOWMARK = INDEX (SUBSTR (BUFF, LASTMARK + 1),'@');
END;
MKS.LOC (J) = LENGTH (BUFF) + 1;
MKS.LIMIT = J;
END SETMKS;
  
```



NEW06660  
 NEW06670  
 NEW06680  
 NEW06690  
 NEW06700  
 NEW06710  
 NEW06720  
 NEW06730  
 NEW06740  
 NEW06750  
 NEW06760  
 NEW06770  
 NEW06780  
 NEW06790  
 NEW06800  
 NEW06810  
 NEW06820  
 NEW06830  
 NEW06840  
 NEW06850  
 NEW06860  
 NEW06870  
 NEW06880  
 NEW06890  
 NEW06900  
 NEW06910  
 NEW06920  
 NEW06930  
 NEW06940  
 NEW06950  
 NEW06960  
 NEW06970  
 NEW06980  
 NEW06990  
 NEW07000  
 NEW07010  
 NEW07020

```

BDISPLAY: PROC (BUFF, STARTLN, NUML, MKS, PRTLNM);
/* THIS PROCEDURE DISPLAYS BUFFER CONTENT */
DCL BUFF CHAR (4000) VAR;
DCL (STARTLN, NUML, NUM, I, SPC, LNLOC) FIXED;
DCL I MKS;
    2 LIMIT FIXED;
    2 LOC (0:100) FIXED;
DCL PRTLNM BIT (1);
DCL PRTLINE CHAR (100) VAR;
LNUM = STARTLN;
DO I = 1 TO NUML;
    IF LNUM < MKS.LIMIT THEN DO;
        LNLOC = MKS.LOC (LNUM) + 1;
        PRTLINE = SUBSTR (BUFF, LNLOC, MKS.LOC (LNUM + 1) - LNLOC);
        SPC = FIXED (GETS (PRTLINE, ':'));
        IF PRTLNM THEN
            PUT SKIP EDIT (LNUM, PRTLINE) (F(2), X (SPC + 1), A);
        ELSE
            PUT SKIP EDIT (PRTLINE) (X (SPC), A);
        END;
    ELSE IF LNUM = MKS.LIMIT THEN DO;
        IF PRTLNM THEN
            PUT SKIP EDIT (LNUM, '***NO MORE***') (F(2), X (23), A);
        ELSE
            PUT SKIP EDIT ('***NO MORE***') (X(25), A);
        END;
    ELSE
        PUT SKIP EDIT ('...')(A);
        LNUM = LNUM + 1;
    END;
END BDISPLAY;
  
```

NEW07030  
 NEW07040  
 NEW07050  
 NEW07060  
 NEW07070  
 NEW07080  
 NEW07090  
 NEW07100  
 NEW07110  
 NEW07120  
 NEW07130  
 NEW07140  
 NEW07150  
 NEW07160  
 NEW07170  
 NEW07180

```

DELTE: PROC;
/* THIS IS THE LINE EDITOR FUNCTION DELETE */
DCL (I,LEN) FIXED;
TRANSMKS = SUBSTR (TRANSMKS,1,TRANSMKS.LOC (LINENM)-1)
      !! SUBSTR (TRANSMKS,TRANSMKS.LOC (LINENM + 1));
LEN = TRANSMKS.LOC (LINENM + 1) - TRANSMKS.LOC (LINENM);
TRANSMKS.LIMIT = TRANSMKS.LIMIT - 1;
DO I = LINENM TO TRANSMKS.LIMIT;
  TRANSMKS.LOC (I) = TRANSMKS.LOC (I+1) - LEN;
END;
END DELTE;

```

```

WTNLM: PROC (ARGSTR,ARGLM,ARGNUM) RETURNS (BIT(1));
/* THIS FUNCTION IS USED TO CHECK VALIDITY OF EDITOR COMMANDS */
DCL ARGSTR CHAR (80) VAR;
DCL (ARGLM,ARGNUM) FIXED;
ON CONVERSION GOTO BAD;
ARGNUM = FIXED (ARGSTR);
IF ARGNUM >= 0 & ARGNUM <= ARGLM THEN
    RETURN ('1'B);
BAD:
RETURN ('0'B);
END WTNLM;

FINISHED:
END NEWBUF;

```

```

NEW07190
NEW07200
NEW07210
NEW07220
NEW07230
NEW07240
NEW07250
NEW07260
NEW07270
NEW07280
NEW07290
NEW07300
NEW07310
NEW07320
NEW07330
NEW07340
NEW07350
NEW07360
NEW07370

```

```

ACT00010
ACT00020
ACT00030
ACT00040
ACT00050
ACT00060
ACT00070
ACT00080
ACT00090
ACT00100
ACT00110
ACT00120
ACT00130
ACT00140
ACT00150
ACT00160
ACT00170
ACT00180
ACT00190
ACT00200
ACT00210
ACT00220
ACT00230
ACT00240
ACT00250
ACT00260
ACT00270
ACT00280
ACT00290
ACT00300
ACT00310
ACT00320
ACT00330
ACT00340
ACT00350
ACT00360
ACT00370
ACT00380
ACT00390
ACT00400
ACT00410
ACT00420
ACT00430
ACT00440
ACT00450

ACTGRD: PROC (UNIT,TKLSPTR,DICTION);
/*****
/* THIS PROGRAM COORDINATES THE ACTIVITIES IN THE VIRTUAL INFORMATION*/ACT00080
/* LEVEL, WHICH IS BELOW THE USER INTERFACE LEVEL.
/* IT CALLS THE TOKENIZER, THE PARSER, THE SIMPLIFIER, THE
/*****
XINCLUDE DICTION;
XINCLUDE XCHANGE;
XINCLUDE ENTITY;
XINCLUDE TOKEN;
XINCLUDE MACH;
XINCLUDE XTREE;
XINCLUDE RETEARG;

DCL UNIT CHAR (2000) VAR;
DCL (TKLSPTR,RETEP) PTR;
DCL TKNIZE ENTRY EXTERNAL;
DCL GO BIT (1) INIT ('1'B);
DCL DEFMCH ENTRY EXTERNAL;
DCL PARSE ENTRY EXTERNAL;
DCL SMPPLY ENTRY EXTERNAL;
DCL GARBAGE CHAR (80);

CALL TKNIZE (UNIT,TKLSPTR,DICTION.GO);
IF ~GO THEN RETURN;
CALL PRINTT;
CALL DEFMCH (MACH);
CALL PARSE (MACH,TKLSPTR,XTREE,XCHANGE,ENTITY,'1'B);
/* LAST BIT PARAMETER CONTROLS THE DEBUG "TRACE" FEATURE */
/* WHICH SHOWS THE STATE TRANSITIONS AND STACK CONTENTS */
/* OF THE PUSH-DOWN AUTOMATA. FINITE STATE MACHIN
CALL PRINTX;
/* CALL PRINT: */
CALL SMPPLY (XTREE,ENTITY,XCHANGE,RETEP);
CALL PRINTR;
PUT EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);
GET EDIT (GARBAGE) (A(80));

```

ACT00460  
 ACT00470  
 ACT00480  
 ACT00490  
 ACT00500  
 ACT00510  
 ACT00520  
 ACT00530  
 ACT00540  
 ACT00550  
 ACT00560  
 ACT00570  
 ACT00580  
 ACT00590  
 ACT00600  
 ACT00610  
 ACT00620  
 ACT00630  
 ACT00640

```
PRINT: PROC; /* DEBUGGING TOOL TO PRINT TOKEN CHAIN */
DCL M PTR;
M = TKLSPTR;
DO WHILE (M ^= NULL());
  PUT SKIP EDIT ('TEST_TOKEN =$', M -> TOKEN.ITEM, '$$',
    'TEST_CLASS =$', M -> TOKEN.CLASS, '$$')
    (A,A,A,SKIP,A,A,A);
  M = M -> TOKEN.NEXT;
END;
END PRINT;
```

ACT00650  
 ACT00660  
 ACT00670  
 ACT00680  
 ACT00690  
 ACT00700  
 ACT00710  
 ACT00720  
 ACT00730  
 ACT00740  
 ACT00750  
 ACT00760  
 ACT00770  
 ACT00780  
 ACT00790  
 ACT00800  
 ACT00810  
 ACT00820  
 ACT00830  
 ACT00840  
 ACT00850  
 ACT00860  
 ACT00870

```

PRINTM: PROC: /* DEBUG TOOL TO TRACE THROUGH TRANSITIONS IN */
/* THE FINITE STATE MACHIN */
DCL I FIXED;

DO I = 1 TO MACH.STATE_MAP (7) - 1;
  PUT SKIP EDIT ('STATE_MAP = ', I,
    'MATCH_STR = ', MACH.MATCH (I),
    'ACTIN_STR = ', MACH.ACTION (I),
    'NXT_STATE = ', MACH.NEXT_STATE (I))
    (A,F(5),2 (SKIP,A,A),SKIP,A,F(5));
  END;

DO I = 1 TO 7;
  PUT SKIP EDIT ('STATE = ', I, ' MAP = ', MACH.STATE_MAP (I))
    (2 (A,F(5)));
  END;
END PRINTM;

```

```

PRINTX: PROC; /* DEBUG TOOL TO PRINT EXECUTION TREE */
DCL I FIXED;
DO I = 1 TO 500 WHILE (XTREE (I).LABEL ^= '');
| XTREE (I).CHILD ^= 0
| XTREE (I).LINK ^= 0);
PUT SKIP EDIT ('XTREE_LOC = ', I,
'LABEL = ', XTREE (I).LABEL,
'CHILD = ', XTREE (I).CHILD,
'LINK = ', XTREE (I).LINK,
(A.F(5), SKIP, A.A.2(SKIP, A.F(5)))));
END;
END PRINTX;

```

```

ACT00880
ACT00890
ACT00900
ACT00910
ACT00920
ACT00930
ACT00940
ACT00950
ACT00960
ACT00970
ACT00980
ACT00990
ACT01000
ACT01010
ACT01020
ACT01030
ACT01040
ACT01050
ACT01060
ACT01070
ACT01080

```

ACT01090  
ACT01100  
ACT01110  
ACT01120  
ACT01130  
ACT01140  
ACT01150  
ACT01160  
ACT01170  
ACT01180  
ACT01190  
ACT01200  
ACT01210  
ACT01220  
ACT01230  
ACT01240  
ACT01250  
ACT01260  
ACT01270  
ACT01280  
ACT01290  
ACT01300  
ACT01310  
ACT01320  
ACT01330  
ACT01340  
ACT01350  
ACT01360  
ACT01370  
ACT01380  
ACT01390  
ACT01400  
ACT01410  
ACT01420  
ACT01430

```

PRINT: PROC; /* DEBUG TOOL TO PRINT ENTITY SET TABLE */
DCL (I,J) FIXED;

DO I = 1 TO 12 WHILE (ENTITY (I).VES_FN ^= '');
  PUT SKIP EDIT ('ENTITY: ',I)(A,F (2));
  PUT SKIP EDIT ('NAME: ',ENTITY (I).NAME)(COL (2),A,A);
  PUT SKIP EDIT ('VES_FN: ',ENTITY (I).VES_FN)(COL (2),A,A);
  PUT SKIP EDIT ('WHERE: ',ENTITY (I).WHERE)(COL (2),A,F (4));
  PUT SKIP EDIT ('N_PARENTS: ',
    ENTITY (I).N_PARENT (1),
    ENTITY (I).N_PARENT (2),
    (COL (2),A,F (2),X (10),F (2)));
  DO J = 1 TO 15 WHILE (ENTITY (I).ATTR (J).USES ^= '');
    PUT SKIP EDIT ('ATTRIBUTE: ',J)(COL (5),A,F (2));
    IF ENTITY (I).ATTR (J).VES_KEY THEN
      PUT SKIP EDIT ('VES_KEY: ',COL (7),A);
    IF ENTITY (I).ATTR (J).CART_KEY THEN
      PUT SKIP EDIT ('CART_KEY: ',COL (7),A);
    PUT SKIP EDIT
      ('A_PARENT: ',ENTITY (I).ATTR (J).A_PARENT)
      (COL (7),A,F (2));
    PUT SKIP EDIT
      ('USES: ',ENTITY (I).ATTR (J).USES)
      (COL (7),A,A);
  END;
END;

END PRINT:;

```



ACT01440  
ACT01450  
ACT01460  
ACT01470  
ACT01480  
ACT01490  
ACT01500  
ACT01510  
ACT01520  
ACT01530  
ACT01540  
ACT01550  
ACT01560  
ACT01570  
ACT01580  
ACT01590  
ACT01600  
ACT01610  
ACT01620  
ACT01630  
ACT01640  
ACT01650  
ACT01660  
ACT01670  
ACT01680  
ACT01690  
ACT01700  
ACT01710  
ACT01720  
ACT01730  
ACT01740  
ACT01750  
ACT01760  
ACT01770  
ACT01780  
ACT01790  
ACT01800  
ACT01810  
ACT01820  
ACT01830  
ACT01840  
ACT01850  
ACT01860  
ACT01870  
ACT01880

PRINTR: PROC; /\* DEBUG TOOL TO PRINT REVISED ENTITY SET TABLE \*/

DCL (I,J) FIXED.

RP PTR;

RP = RETEP;

DO WHILE (RP ^= NULL ());

PUT SKIP EDIT

('ENT\_NAME: ',RP -> RETE\_ARG.ENT\_NAME,

'ENT\_DEPTH: ',RP -> RETE\_ARG.ENT\_DEPTH)

(A.A.SKIP,A.F(5));

DO I = 1 TO RP -> RETE\_ARG.NUM\_ATTR;

PUT SKIP EDIT ('ATTRIBUTE: ',I)(A.F(3));

IF RP -> RETE\_ARG.ENT\_ATTR (I).SING OCC THEN

PUT SKIP EDIT ('SING\_OCC')(COL(5),A);

PUT SKIP EDIT

('A\_PARENT: ',RP -> RETE\_ARG.ENT\_ATTR (I).A\_PARENT,

'USES: ',RP -> RETE\_ARG.ENT\_ATTR (I).USES)

(COL(5),A.A.SKIP.COL(5),A.A);

END;

DO I = 1 TO RP -> RETE\_ARG.NUM\_COND;

PUT SKIP EDIT

('COND: ',I,

'ATTRREF: ',RP -> RETE\_ARG.COND (I).ATTRREF)

(A.F(3),SKIP.COL(2),A.F(3));

IF RP -> RETE\_ARG.COND (I).NEG THEN

PUT SKIP EDIT ('NEGATE')(COL(2),A);

PUT SKIP EDIT ('REL: ',RP -> RETE\_ARG.COND (I).REL)(COL(2),A.A);

DO J = 1 TO 10 WHILE (RP -> RETE\_ARG.COND (I).CDATA (J) ^= '');

PUT SKIP EDIT

(J,RP -> RETE\_ARG.COND (I).CDATA (J))

(COL(5),F(3),X(1),A);

END;

END;

RP = RP -> RETE\_ARG.CTL\_INFO.PTR;

END;

END PRINTR;

GETS: PROC (LIST,TERM\_ITEM) RETURNS (CHAR (30) VAR):

DCL LIST CHAR (\*) VAR,  
TERM\_ITEM CHAR (1),  
RTN\_LIST CHAR (30) VAR,  
I FIXED;

I = INDEX (LIST,TERM\_ITEM):

IF I = 0 THEN DO;

RTN\_LIST = LIST;

LIST = ',';

END;

ELSE DO;

RTN\_LIST = SUBSTR (LIST,I,I - 1);

LIST = SUBSTR (LIST,I + 1);

END;

RETURN (RTN\_LIST);

END GETS;

ACT01890  
ACT01900  
ACT01910  
ACT01920  
ACT01930  
ACT01940  
ACT01950  
ACT01960  
ACT01970  
ACT01980  
ACT01990  
ACT02000  
ACT02010  
ACT02020  
ACT02030  
ACT02040  
ACT02050  
ACT02060  
ACT02070  
ACT02080  
ACT02090  
ACT02100  
ACT02110  
ACT02120  
ACT02130

ACT02140

END ACTCRD:

```

/* THIS PROCEDURE TOKENIZES INPUT RETRIEVE STATEMENTS;
EXECUTES DEFINE, ADHOC, AND LISTDEF STATEMENTS, AND
REPLACES VIRTUAL DEFINITION NAMES BY THEIR CORRESPONDING
VIRTUAL DEFINITIONS.
*/
TKNIZE: PROC (UNIT, TKLSPTR, DICTION, GO);
  %INCLUDE DICTION;
  %INCLUDE TOKEN;
  DCL DCTNRY ENTRY (, CHAR (160) VAR)
    EXTERNAL RETURNS (CHAR (160) VAR);
  DCL UNIT CHAR (2000) VAR;
  DCL GO BIT (1);
  DCL TKLSPTR PTR;
  DCL WORD CHAR (80) VAR;
  DCL PLIST1(3) CHAR(8) INIT ('DVHJTL', 'TEST', HIGH(8)),
  DCL PLIST2(3) CHAR(8) INIT ('DVHJTL', 'CLEAR', HIGH(8)),
  (HIGH, PLIRETC) BUILTIN;
  RETCODE FIXED BINARY (31,0);
  CMSGMD EXTERNAL ENTRY OPTIONS (ASSEMBLER INTER);
  DCL GARBAGE CHAR (2000) VAR;
  DCL KIND FIXED INIT (0);
  CALL CMSGMD (PLIST2, RETCODE);
  WORD = NTKSTR (UNIT, KIND);
  SELECT (WORD);
    WHEN ('DEFINE', 'DEF', 'ADHOC')
      CALL DEF (UNIT, TKLSPTR, KIND, WORD);
    WHEN (',',) DO;
      UNIT = '';
      GO = 'O'B;
      RETURN;
    END;
    WHEN ('RETRIEVE', 'RET')
      UNIT = 'RETRIEVE', ; UNIT;
    WHEN ('LISTDEF')
      CALL LISTDEF (UNIT, TKLSPTR);
    OTHERWISE DO;
      CALL MSG ('UNRECOGNIZED COMMAND--', ; WORD);
      UNIT = '';
      TKLSPTR = NULL();
      RETURN;
    END;
  END;

CALL BDTKCHN (UNIT, TKLSPTR, WORD, KIND);

```

TKN00010  
 TKN00020  
 TKN00030  
 TKN00040  
 TKN00050  
 TKN00060  
 TKN00070  
 TKN00080  
 TKN00090  
 TKN00100  
 TKN00110  
 TKN00120  
 TKN00130  
 TKN00140  
 TKN00150  
 TKN00160  
 TKN00170  
 TKN00180  
 TKN00190  
 TKN00200  
 TKN00210  
 TKN00220  
 TKN00230  
 TKN00240  
 TKN00250  
 TKN00260  
 TKN00270  
 TKN00280  
 TKN00290  
 TKN00300  
 TKN00310  
 TKN00320  
 TKN00330  
 TKN00340  
 TKN00350  
 TKN00360  
 TKN00370  
 TKN00380  
 TKN00390  
 TKN00400  
 TKN00410  
 TKN00420  
 TKN00430  
 TKN00440  
 TKN00450  
 TKN00460  
 TKN00470  
 TKN00480  
 TKN00490  
 TKN00500

```

GETS: PROC (LIST,TERM_ITEM) RETURNS (CHAR (1000) VAR);
DCL LIST CHAR (*) VAR,
     TERM_ITEM CHAR (1),
     RTN_LIST CHAR (1000) VAR,
     I FIXED;
I = INDEX (LIST,TERM_ITEM);
IF I = 0 THEN DO;
  RTN_LIST = LIST;
  LIST = ',';
END;
ELSE DO;
  RTN_LIST = SUBSTR (LIST,I - 1);
  LIST = SUBSTR (LIST,I + 1);
END;
RETURN (RTN_LIST);
END GETS;

```

```

TKN00510
TKN00520
TKN00530
TKN00540
TKN00550
TKN00560
TKN00570
TKN00580
TKN00590
TKN00600
TKN00610
TKN00620
TKN00630
TKN00640
TKN00650
TKN00660
TKN00670
TKN00680
TKN00690
TKN00700
TKN00710
TKN00720
TKN00730
TKN00740

```

TKNO0750  
TKNO0760  
TKNO0770  
TKNO0780  
TKNO0790  
TKNO0800  
TKNO0810  
TKNO0820  
TKNO0830  
TKNO0840  
TKNO0850  
TKNO0860  
TKNO0870  
TKNO0880  
TKNO0890  
TKNO0900  
TKNO0910  
TKNO0920  
TKNO0930  
TKNO0940  
TKNO0950  
TKNO0960  
TKNO0970  
TKNO0980  
TKNO0990  
TKNO1000  
TKNO1010

```

NXTKSTR: PROC (UNIT,KIND) RETURNS (CHAR (80) VAR):
DCL UNIT CHAR (2000) VAR;
DCL SYMBOL CHAR (1);
DCL KIND FIXED;
KIND = 0;
IF UNIT = '' THEN RETURN ('');
DO WHILE ('1'B):
  CALL RMVFLKS (UNIT);
  IF UNIT = '' THEN RETURN ('');
  IF INDEX ('\@',SUBSTR (UNIT,1,1)) ^= 0 THEN
    DO WHILE (INDEX ('\@',SUBSTR (UNIT,1,1)) ^= 0):
      SYMBOL = SUBSTR (UNIT,1,1);
      UNIT = SUBSTR (UNIT,2);
      IF SYMBOL = '\ ' THEN
        GARBAGE = GETS (UNIT,SYMBOL);
      ELSE
        GARBAGE = GETS (UNIT,' ');
      END;
    IF SUBSTR (UNIT,1,1) ^= ' ' THEN
      RETURN (TOK1 (UNIT,KIND));
    END;
  
```

TKNO1020  
TKNO1030  
TKNO1040  
TKNO1050  
TKNO1060  
TKNO1070  
TKNO1080  
TKNO1090  
TKNO1100  
TKNO1110  
TKNO1120  
TKNO1130  
TKNO1140  
TKNO1150  
TKNO1160  
TKNO1170  
TKNO1180  
TKNO1190

```

RMVFBKLS: PROC (STRING);
DCL STRING CHAR (2000) VAR;
DCL I FIXED;
IF STRING = '' THEN RETURN;
DO I = 1 TO LENGTH (STRING);
  IF SUBSTR (STRING,I,1) ~=' ' , THEN DO;
    STRING = SUBSTR (STRING, I);
  RETURN;
  END;
END;
STRING = '' ;
RETURN;
END RMVFBKLS;
  
```

/\* REMOVE FRONT BLANKS \*/

TKNO 1200  
TKNO 1210  
TKNO 1220  
TKNO 1230  
TKNO 1240  
TKNO 1250  
TKNO 1260  
TKNO 1270  
TKNO 1280  
TKNO 1290  
TKNO 1300  
TKNO 1310  
TKNO 1320  
TKNO 1330  
TKNO 1340  
TKNO 1350  
TKNO 1360  
TKNO 1370  
TKNO 1380  
TKNO 1390  
TKNO 1400  
TKNO 1410  
TKNO 1420  
TKNO 1430  
TKNO 1440  
TKNO 1450  
TKNO 1460  
TKNO 1470  
TKNO 1480  
TKNO 1490  
TKNO 1500  
TKNO 1510  
TKNO 1520  
TKNO 1530  
TKNO 1540  
TKNO 1550  
TKNO 1560  
TKNO 1570  
TKNO 1580  
TKNO 1590  
TKNO 1600  
TKNO 1610  
TKNO 1620  
TKNO 1630  
TKNO 1640  
TKNO 1650  
TKNO 1660  
TKNO 1670  
TKNO 1680  
TKNO 1690  
TKNO 1700  
TKNO 1710  
TKNO 1720  
TKNO 1730  
TKNO 1740  
TKNO 1750  
TKNO 1760  
TKNO 1770

```
TOK1: PROC (UNIT,KIND) RETURNS (CHAR (80) VAR);
DOCL (UNIT,CJTCLAS) CHAR (2000) VAR;
DOCL (KIND,I,U) FIXED;
DOCL TKSTR CHAR (80) VAR;
DOCL (CLASS,PREFCLASS) CHAR (1);
IF UNIT = '' THEN RETURN ('');
UNCLAS = TRANSLATE (UNIT,
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
'ABCDEFGHIJKLMNPOQRSTUVWXYZO123456789.-/+!*|^><=>);
TKSTR = '';
PREFCLASS = 'B';
DOO I = 1 TO LENGTH (UNIT);
CLASS = SUBSTR (UNITCLAS,I,1);
SELECT (PREFCLASS);
WHEN ('B','M','O') DO;
TKSTR = SUBSTR (UNIT,I,1);
IF CLASS = 'B' ; CLASS = 'M' ; CLASS =
IF I = LENGTH (UNIT) THEN DO;
UNIT = '';
RETURN (TKSTR);
END;
ELSE DO;
UNIT = SUBSTR (UNIT,I+1);
RETURN (TKSTR);
END;
PRECLASS = CLASS;
IF I = LENGTH (UNIT) THEN DO;
UNIT = '';
IF CLASS = 'N' THEN
KIND = 1;
RETURN (TKSTR);
END;
END;
WHEN ('A')
IF INDEX ('ANOM',CLASS) ~= 0 THEN DO;
TKSTR = TKSTR || SUBSTR (UNIT,I,1);
IF I = LENGTH (UNIT) THEN DO;
UNIT = '';
RETURN (TKSTR);
END;
END;
ELSE DO;
UNIT = SUBSTR (UNIT,I);
RETURN (TKSTR);
END;
WHEN ('N')
IF INDEX ('ND',CLASS) ~= 0 THEN DO;
TKSTR = TKSTR || SUBSTR (UNIT,I,1);
IF I = LENGTH (UNIT) THEN DO;
UNIT = '';
KIND = 1;
RETURN (TKSTR);
END;
```



TKNO1780  
TKNO1790  
TKNO1800  
TKNO1810  
TKNO1820  
TKNO1830  
TKNO1840  
TKNO1850  
TKNO1860  
TKNO1870  
TKNO1880  
TKNO1890  
TKNO1900  
TKNO1910  
TKNO1920  
TKNO1930  
TKNO1940  
TKNO1950  
TKNO1960  
TKNO1970  
TKNO1980  
TKNO1990  
TKNO2000  
TKNO2010  
TKNO2020  
TKNO2030  
TKNO2040  
TKNO2050  
TKNO2060  
TKNO2070  
TKNO2080  
TKNO2090  
TKNO2100  
TKNO2110  
TKNO2120  
TKNO2130  
TKNO2140  
TKNO2150  
TKNO2160  
TKNO2170  
TKNO2180  
TKNO2190  
TKNO2200  
TKNO2210  
TKNO2220  
TKNO2230  
TKNO2240  
TKNO2250  
TKNO2260  
TKNO2270  
TKNO2280

```

END:
ELSE DO:
  KIND = 1;
  UNIT = SUBSTR (UNIT,I);
  RETURN (TKSTR);
END:
WHEN ('D')
  IF CLASS = 'N' THEN DO:
    TKSTR = TKSTR || SUBSTR (UNIT,I,1);
    IF I = LENGTH (UNIT) THEN DO:
      KIND = 1;
      UNIT = '';
      RETURN (TKSTR);
    END:
  ELSE DO:
    UNIT = SUBSTR (UNIT,I);
    KIND = 1;
    RETURN (TKSTR);
  END:
WHEN ('O') DO:
  J = INDEX (SUBSTR (UNITCLAS,2),'O');
  IF J > 0 THEN
    TKSTR = SUBSTR (UNIT,2,J-1);
  ELSE
    TKSTR = UNIT;
  IF J = LENGTH (UNIT) - 1 THEN
    UNIT = '';
  ELSE
    UNIT = SUBSTR (UNIT,J+2);
  KIND = 2;
  RETURN (TKSTR);
END:
WHEN ('S') DO:
  TKSTR = TKSTR || SUBSTR (UNIT,I,1) ;
  IF I = LENGTH (UNIT) THEN
    UNIT = '';
  ELSE
    UNIT = SUBSTR (UNIT,I+1);
  RETURN (TKSTR);
END:
OTHERWISE:
END:
END TOK1:
END NXTKSTR:

```

TKNO2290  
TKNO2300  
TKNO2310  
TKNO2320  
TKNO2330  
TKNO2340  
TKNO2350  
TKNO2360  
TKNO2370  
TKNO2380  
TKNO2390  
TKNO2400  
TKNO2410  
TKNO2420  
TKNO2430  
TKNO2440  
TKNO2450  
TKNO2460  
TKNO2470  
TKNO2480  
TKNO2490  
TKNO2500  
TKNO2510  
TKNO2520  
TKNO2530  
TKNO2540  
TKNO2550  
TKNO2560  
TKNO2570  
TKNO2580  
TKNO2590  
TKNO2600  
TKNO2610  
TKNO2620  
TKNO2630  
TKNO2640  
TKNO2650  
TKNO2660  
TKNO2670  
TKNO2680  
TKNO2690  
TKNO2700  
TKNO2710  
TKNO2720  
TKNO2730  
TKNO2740  
TKNO2750  
TKNO2760  
TKNO2770  
TKNO2780  
TKNO2790  
TKNO2800  
TKNO2810  
TKNO2820

```

/* * THIS PROCEDURE DEFINES AND REMOVES DEFINITIONS FROM
ADHOC AND/OR PERMANENT DICTIONARIES
*/
DEF: PROC (UNIT, TKLSPTR, KIND, WORD);
DOCL UNIT CHAR (2000) VAR;
DOCL TKLSPTR PTR;
DOCL KIND FIXED;
DOCL (WORD, WORDO, VNAME, VNAME1) CHAR (80) VAR;
DOCL WORD1 CHAR (160) VAR;
DOCL DEFMSG CHAR (160) VAR;
IF WORD = 'DEF' THEN WORD = 'DEFINE';
WORDO = WORD;
VNAME = NTKSTR (UNIT, KIND);
VNAME1 = TRANSLATE (VNAME, '0123456789#%&_#{}?.,''');
IF SUBSTR (VNAME1, 1, 1) = '#' THEN DO;
CALL MSG ('@', VNAME, 'DEFINITION: *ERROR* INVALID NAME');
UNIT = '';
TKLSPTR = NULL();
RETURN;
END;
WORD = NTKSTR (UNIT, KIND);
IF WORD ^= 'AS' THEN
IF ~ (WORD = 'REMOVE' | WORD = 'REM') THEN DO;
UNIT = '';
TKLSPTR = NULL();
CALL MSG ('MISSING "AS" IN DEFINE OR ADHOC STMT');
RETURN;
END;
ELSE DO;
IF WORDO = 'DEFINE' THEN
DEFMSG = DCTNRY (DICTION, 'REMOVE', VNAME);
ELSE
DEFMSG = DCTNRY (DICTION, 'ADHREMOVE', VNAME);
UNIT = '';
CALL MSG ('@', VNAME, '@ REMOVE: ' || GETS (DEFMSG, ''));
END;
ELSE DO;
TKLSPTR = NULL();
WORD1 = GETS (UNIT, '');
IF WORDO = 'DEFINE' THEN
WORD1 = 'SAVE', VNAME, '|| WORD1;
ELSE
WORD1 = 'ADHSAVE', VNAME, '|| WORD1;
DEFMSG = DCTNRY (DICTION, WORD1);
CALL MSG ('@', VNAME, '@ DEFINITION: ' || GETS (DEFMSG, ''));
END;
GO = 'O'B;
END DEF;

```

TKNO2830  
TKNO2840  
TKNO2850  
TKNO2860  
TKNO2870  
TKNO2880  
TKNO2890  
TKNO2900  
TKNO2910  
TKNO2920  
TKNO2930  
TKNO2940  
TKNO2950  
TKNO2960  
TKNO2970  
TKNO2980  
TKNO2990  
TKNO3000  
TKNO3010  
TKNO3020  
TKNO3030  
TKNO3040  
TKNO3050  
TKNO3060  
TKNO3070  
TKNO3080  
TKNO3090  
TKNO3100  
TKNO3110  
TKNO3120  
TKNO3130  
TKNO3140  
TKNO3150  
TKNO3160  
TKNO3170  
TKNO3180  
TKNO3190  
TKNO3200  
TKNO3210  
TKNO3220  
TKNO3230  
TKNO3240  
TKNO3250  
TKNO3260  
TKNO3270  
TKNO3280  
TKNO3290  
TKNO3300  
TKNO3310  
TKNO3320  
TKNO3330  
TKNO3340  
TKNO3350  
TKNO3360  
TKNO3370  
TKNO3380  
TKNO3390  
TKNO3400

```
BDTKCHN: PROC (UNIT,TKLSPTR,WORD,KIND); /* BUILDS TOKEN CHAIN */
DCL UNIT CHAR (2000) VAR;
DCL (TKLSPTR,P,TAIL) PTR;
DCL WORD CHAR (80) VAR;
DCL (FINDMSG,CPFMMSG,SUBSTITUTE) CHAR (160) VAR;
DCL KIND FIXED;
DCL WORD1 CHAR (160) VAR;
DO WHILE (UNIT ^= '');
    WORD = NXTKSTR (UNIT,KIND);
    IF KIND ^= 0 THEN DO;
        ALLOCATE TOKEN SET (P);
        IF KIND = 1 THEN DO;
            P -> TOKEN.CLASS = 'A';
            P -> TOKEN.ITEM = CHAR (FIXED (WORD));
        ELSE DO;
            P -> TOKEN.CLASS = 'S';
            P -> TOKEN.ITEM = WORD;
        END;
        P -> TOKEN.NEXT = NULL ();
        IF TKLSPTR = NULL () THEN
            TKLSPTR = P;
        ELSE
            TAIL -> TOKEN.NEXT = P;
        TAIL = P;
    END;
ELSE DO;
    WORD1 = 'FIND.' || WORD;
    FINDMSG = DCTNRY (DICTION,WORD1);
    CPFMSG = FINDMSG;
    SUBSTITUTE = GETS (CPFMSG,'');
    IF CPFMSG ^= 'V' THEN DO;
        IF SUBSTR (SUBSTITUTE,1,1) = '1' THEN DO;
            CALL MSG (WORD || ' ' || SUBSTR (SUBSTITUTE,2));
            UNIT = '';
            TKLSPTR = NULL ();
            RETURN;
        END;
        ALLOCATE TOKEN SET (P);
        P -> TOKEN.ITEM = GETS (FINDMSG,'');
        P -> TOKEN.CLASS = FINDMSG;
        P -> TOKEN.NEXT = NULL ();
        IF TKLSPTR = NULL () THEN
            TKLSPTR = P;
        ELSE
            TAIL -> TOKEN.NEXT = P;
        TAIL = P;
    END;
ELSE
    UNIT = SUBSTITUTE || ' ' || UNIT;
END;
END BDTKCHN;
```

TKN03410  
TKN03420  
TKN03430  
TKN03440  
TKN03450  
TKN03460  
TKN03470  
TKN03480  
TKN03490  
TKN03500  
TKN03510  
TKN03520  
TKN03530  
TKN03540  
TKN03550

```

/* THIS PROCEDURE OUTPUTS ARBITRARY MESSAGES AND WAITS
   FOR THE USER TO TYPE THE "ENTER" KEY TO CONTINUE */
MSG: PROC(LINE);
  DCL LINE CHAR (*) VAR;
  CALL CMSCMD (PLIST2, RETCODE);
  PUT SKIP EDIT (LINE) (A);
  PUT SKIP EDIT ('TYPE "ENTER" KEY TO CONTINUE') (A);
  GET EDIT (GARBAGE) (A(80));
  GO - 'O'B;
END MSG;

```

```

/* THIS PROCEDURE OUTPUTS THE STORED DEFINITIONS */
LISTDEF: PROC (UNIT,TKLSPTR);
DCL UNIT CHAR (2000) VAR;
DCL (DEFMSG,DEFMSG1) CHAR (160) VAR;
DCL WORD CHAR (80) VAR;
DCL TKLSPTR PTR;
DCL KIND FIXED INIT (0);
UNIT = SUBSTR (UNIT,1,INDEX (UNIT,':')-1);
WORD = NTKSTR (UNIT,KIND);
UNIT = '';
TKLSPTR = NULL();
IF WORD = '' || KIND = 1 || KIND = 2 THEN DO;
CALL MSG ('*ERROR* INVALID ARGUMENT TO "LISTDEF" STATEMENT');
RETURN;
END;
DEFMSG = DCTNRY (DICTION,'FIND',||WORD);
DEFMSG1 = GETS (DEFMSG,');');
IF DEFMSG ^= ':V' THEN DO;
CALL MSG ('*ERROR* NO DEFINITION STORED FOR *' || WORD || '*');
RETURN;
END;
CALL CMSCMD (PLIST2.RETCODE);
PUT SKIP EDIT
('DEFINITION OF ' || WORD || ': ' ) (A(80));
CALL DEFDSPLY (DEFMSG1);
GET EDIT (GARBAGE) (A(80));
GO = 'O'B;
RETURN;

```

TKN03560  
 TKN03570  
 TKN03580  
 TKN03590  
 TKN03600  
 TKN03610  
 TKN03620  
 TKN03630  
 TKN03640  
 TKN03650  
 TKN03660  
 TKN03670  
 TKN03680  
 TKN03690  
 TKN03700  
 TKN03710  
 TKN03720  
 TKN03730  
 TKN03740  
 TKN03750  
 TKN03760  
 TKN03770  
 TKN03780  
 TKN03790  
 TKN03800  
 TKN03810  
 TKN03820  
 TKN03830  
 TKN03840  
 TKN03850  
 TKN03860  
 TKN03870  
 TKN03880

```

/* THIS PROCEDURE IS CALLED BY LISTDEF TO TRANSLATE
PREVIOUSLY TRANSLATED CHARACTERS WITHIN THE VIRTUAL
DEFINITIONS, AND THEN TO DISPLAY THE STORED
VIRTUAL DEFINITIONS.
*/

```

```

DEFSPLY: PROC (DFNTN);
DCL DFNTN CHAR (160) VAR;
DCL LINE CHAR (160) VAR;
DCL I FIXED;
DCL SPACES FIXED INIT (0);
DCL SPACECH CHAR (2) VAR;
DO WHILE (DFNTN ^= '');
  LINE = GETS (DFNTN, ' ');
  IF LINE ^= '' THEN DO;
    CALL REPLACE (LINE, '%1', ' ');
    CALL REPLACE (LINE, '%2', ' ');
    CALL REPLACE (LINE, '%3', ' ');
    CALL REPLACE (LINE, '%5', ' ');
    CALL REPLACE (LINE, '%4', '%');
    IF SPACES ^= 0 THEN
      DO I = 1 TO SPACES;
        LINE = ' ' || LINE;
      END;
    PUT EDIT (LINE) (COL(1), A(80));
    IF DFNTN ^= '' THEN
      DFNTN = ' ' || DFNTN;
    END;
  ELSE DO;
    SPACECH = GETS (DFNTN, ' ');
    SPACES = FIXED (SPACECH);
    END;
  END;
END DEFSPLY;
END LISTDEF;

```

TKNO3890  
 TKNO3900  
 TKNO3910  
 TKNO3920  
 TKNO3930  
 TKNO3940  
 TKNO3950  
 TKNO3960  
 TKNO3970  
 TKNO3980  
 TKNO3990  
 TKNO4000  
 TKNO4010  
 TKNO4020  
 TKNO4030  
 TKNO4040  
 TKNO4050  
 TKNO4060  
 TKNO4070  
 TKNO4080  
 TKNO4090  
 TKNO4100  
 TKNO4110  
 TKNO4120  
 TKNO4130  
 TKNO4140  
 TKNO4150  
 TKNO4160  
 TKNO4170  
 TKNO4180  
 TKNO4190  
 TKNO4200  
 TKNO4210  
 TKNO4220  
 TKNO4230  
 TKNO4240  
 TKNO4250  
 TKNO4260  
 TKNO4270

```

/* THIS PROCEDURE REPLACES CERTAIN CHARACTERS BY OTHER
   PRESCRIBED CHARACTERS
   REPLACE: PROC (RLINE,FR_SYM,TO_SYM);
   DCL (RLINE,REP_LINE) CHAR (160) VAR;
   FR_SYM CHAR (2) VAR;
   TO_SYM CHAR (2) VAR;
   I FIXED;

   REP_LINE = '';
   I = INDEX (RLINE,FR_SYM);
   DO WHILE (I ^= 0);
     IF LENGTH (FR_SYM) = 1 THEN DO;
       REP_LINE = REP_LINE || SUBSTR (RLINE,I-1) || TO_SYM;
       RLINE = SUBSTR (RLINE,I+1);
       I = INDEX (RLINE,FR_SYM);
     END;
     ELSE DO;
       REP_LINE = REP_LINE || SUBSTR (RLINE,I-1) || TO_SYM;
       RLINE = SUBSTR (RLINE,I+2);
       I = INDEX (RLINE,FR_SYM);
     END;
     RLINE = REP_LINE || RLINE;
   END REPLACE;

   FINISHED;
   END TKNIZE;

```

TKN04280  
 TKN04290  
 TKN04300  
 TKN04310  
 TKN04320  
 TKN04330  
 TKN04340  
 TKN04350  
 TKN04360  
 TKN04370  
 TKN04380  
 TKN04390  
 TKN04400  
 TKN04410  
 TKN04420  
 TKN04430  
 TKN04440  
 TKN04450  
 TKN04460  
 TKN04470  
 TKN04480  
 TKN04490  
 TKN04500  
 TKN04510  
 TKN04520  
 TKN04530  
 TKN04540  
 TKN04550  
 TKN04560  
 TKN04570  
 TKN04580  
 TKN04590  
 TKN04600  
 TKN04610

# DATA STRUCTURES

```

DCL 1 MACH,
  2 STATE_MAP (200) FIXED,
  2 MATCH (500) CHAR (30) VAR,
  2 ACTION (500) CHAR (40) VAR,
  2 NEXT_STATE (500) FIXED;

DCL 1 TOKEN_BASED,
  2 ITEM_CHAR (30) VAR,
  2 CLASS_CHAR (10) VAR,
  2 NEXT_PTR;

DCL 1 XTREE (1000),
  2 LABEL_CHAR (30) VAR,
  2 CHILD_FIXED,
  2 LINK_FIXED;

DCL 1 ATTRIB_BASED,
  2 LEVEL_FIXED,
  2 ITEM_CHAR (50) VAR,
  2 NEXT_PTR;

DCL 1 ENTITY (12),
  2 NAME_CHAR (30) VAR,
  2 DEPTH_FIXED,
  2 VES_FN_CHAR (2) VAR,
  2 WHERE_FIXED,
  2 N_PARENT (2) FIXED,
  2 VES_PAR_PTR,
  2 ATTR (15),
    3 VES_KEY_BIT (1),
    3 CART_KEY_BIT (1),
    3 SING_OCC_BIT (1),
    3 A_PARENT_FIXED,
    3 USES_CHAR (30) VAR,
    3 LIST_PTR;

DCL 1 N_MAP (2) BASED,
  2 NUM (15) FIXED;

```

```

ALLO0020
ALLO0030
ALLO0040
ALLO0050
ALLO0060
ALLO0070
ALLO0080
ALLO0090
ALLO0100
ALLO0110
ALLO0120
ALLO0130
ALLO0140
ALLO0150
ALLO0160
ALLO0170
ALLO0180
ALLO0190
ALLO0200
ALLO0210
ALLO0220
ALLO0230
ALLO0240
ALLO0250
ALLO0260
ALLO0270
ALLO0280
ALLO0290
ALLO0300
ALLO0310
ALLO0320
ALLO0330
ALLO0340
ALLO0350
ALLO0360
ALLO0370
ALLO0380
ALLO0390
ALLO0400

```



```

DCL 1 DICTION.
  2 CHECK.
    3 LIMIT FIXED.
    3 LABEL (50) CHAR (50) VAR.
    3 TIMES (50) FIXED.
  2 MAIN.
    3 LIMIT FIXED.
    3 FROM (100) CHAR (50) VAR.
    3 TO (100) CHAR (160) VAR.
  2 ADHOC.
    3 LIMIT FIXED.
    3 FROM (50) CHAR (50) VAR.
    3 TO (50) CHAR (160) VAR.

DCL 1 ENTCOND (12,12).
  2 ATTRREF FIXED.
  2 NEG BIT (1).
  2 REL CHAR (1) VAR.
  2 CDATA (10) CHAR (30) VAR.
  2 DORYXGRBEE (20) CHAR (80) VAR.

DCL 1 RETE_ARG BASED. /* WILL ALSO BE RETE_RTN */
  2 CTL_INFO.
    3 LEN FIXED BIN (15).
    3 CBTP FIXED BIN (15) INIT (21).
    3 PTR PTR.
  2 NUM_ATTR FIXED.
  2 NUM_COND FIXED. /* NUMBER OF CONDITIONS */
  2 ENT.
    3 NAME CHAR (30) VAR. /* ENTITY SET NAME */
    3 DEPTH FIXED. /* FILLED IN WHEN RETURNED */
    3 ATTR (NATTR REFER (RETE_ARG.NUM_ATTR)).
    4 SING_OCC BIT (1). /* IF SINGLE OCCUR THEN SET */
    4 A_PARENT FIXED. /* PARENT NUMBER */
    4 USES CHAR(30) VAR. /* ATTRIBUTE NAME */
    4 LIST PTR. /* POINT TO LIST OF OCC OF THIS ATTR IF ANY */
  2 COND (NCOND REFER (RETE_ARG.NUM_COND)).
  3 ATTRREF FIXED. /* POINTER TO ATTR IN ATTR ARRAY ABOVE */
  3 NEG BIT (1). /* DOES A NEGATION ON RELATOR */
  3 REL CHAR (1). /* '<', '>', '>' */
  3 CDATA (10) CHAR (30) VAR. /* UP TO 10 "MULTI" ITEMS */

```

```

ALLO0410
ALLO0420
ALLO0430
ALLO0440
ALLO0450
ALLO0460
ALLO0470
ALLO0480
ALLO0490
ALLO0500
ALLO0510
ALLO0520
ALLO0530
ALLO0540
ALLO0550
ALLO0560
ALLO0570
ALLO0580
ALLO0590
ALLO0600
ALLO0610
ALLO0620
ALLO0630
ALLO0640
ALLO0650

ALLO0670
ALLO0680
ALLO0690
ALLO0700
ALLO0710
ALLO0720
ALLO0730
ALLO0740
ALLO0750
ALLO0760
ALLO0770
ALLO0780
ALLO0790
ALLO0800
ALLO0810
ALLO0820
ALLO0830
ALLO0840
ALLO0850
ALLO0860

```

ALLO0870  
 ALLO0880  
 ALLO0890  
 ALLO0900  
 ALLO0910  
 ALLO0920  
 ALLO0930  
 ALLO0940  
 ALLO0950  
 ALLO0960  
 ALLO0970  
 ALLO0980  
 ALLO0990

DCL 1 RETE\_RTN1 BASED. /\* USED WHEN RETURNED \*/  
 2 CTL.  
 3 LEN FIXED BIN(31).  
 3 CBTP FIXED BIN (31) INIT (46).  
 3 PTR PTR.  
 2 LEVEL FIXED. /\* OCCUR NUMBER \*/  
 2 ITEM CHAR (50) VAR;

# FINITE-STATE-MACHINE RULES MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 1		
1. 1 RETRIEVE		47
STATE NUMBER: 2		0
2. 1 :.RETRIEVE		140
2. 2 BY		
STATE NUMBER: 3		
STATE NUMBER: 11		
11. 1 ~		18
11. 2 (		12
11. 3 :A: S		12
11. 4 -		16
11. 5 +		11
11. 6 :R		14
11. 7 STR		22
11. 8 DATE		12
11. 9 :B		20
STATE NUMBER: 12		
12. 1 )..(		12
12. 2 :.-P		12
12. 3 !		13
12. 4 @SUMOP, .@SUMOP>		12
12. 5 @SUMOP>, .@CONC		13
12. 6 @MULTOP, .@MULTOP>		12
12. 7 @MULTOP, .@SUMOP		13
12. 8 @CONC, .@CONC>		12
12. 9 @CONC>		13
12. 10 :.SUBR		-1
12. 11		12
STATE NUMBER: 13		
13. 1 :R		14
13. 2 :A: S		12
13. 3 -		16
13. 4 +		13
13. 5 (		13
13. 6 :B		17
STATE NUMBER: 14		
14. 1 (		15
14. 2		12
STATE NUMBER: 15		
15. 1 :R		19
STATE NUMBER: 16		
16. 1 -		16
16. 2 +		16
16. 3 :A: S		12
16. 4 :R		14
PUSH, 2, 1@DEL PUSH, 2, SUBR: 2		
DEL PUSH, 2, SUBR: 2		
DEL		
PUSH, 2, 1@DEL		
PUSH, 1, @DEL		
PUSH, 2, 1@: 1 DEL		
DEL		
PUSH, 1, @DEL		
PUSH, 2, SUBR: 12 PUSH, 2, 1@: 6 DEL		
PUSH, 2, 1@: 0 GENNODE DEL		
PUSH, 2, SUBR: 12 PUSH, 2, 1@: 1 DEL		
DEL POP, 2		
GENNODE		
PUSH, 2, 1@: 2 DEL		
GENNODE		
PUSH, 2, 1@: 2 DEL		
GENNODE		
PUSH, 2, 1@: 2 DEL		
GENNODE		
PUSH, 2, 1@: 2 DEL		
GENNODE		
PUSH, 1, @DEL		
PUSH, 1, @DEL		
PUSH, 2, 1@: 1 DEL		
DEL		
PUSH, 2, 1@DEL		
PUSH, 2, 1@: 1 DEL		
PUSH, 2, 1@DEL INDX, 1		
PUSH, 1, 1@DEL		
PUSH, 2, 1@: 1 DEL		
DEL		
PUSH, 1, @DEL		
PUSH, 1, @DEL		

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 17			
17. 1 (	PUSH, 2, 1, 0, DEL	11	
STATE NUMBER: 18			
18. 1 :R	PUSH, 1, C, :C, DEL	14	
STATE NUMBER: 19			
19. 1 (	PUSH, 2, 1, 0, DEL, ADDON, 1, 0, POP, 1	15	
19. 2 ), (	ADDON, 1, 0, POP, 1, DEL, POP, 2	28	
STATE NUMBER: 20			
20. 1 (	DEL, PUSH, 2, SUBR: 21	11	
STATE NUMBER: 21			
21. 1 ..SUBR	DEL, GENNODE	-1	
21. 2 )		21	
STATE NUMBER: 22			
22. 1 (	DEL, PUSH, 2, SUBR: 23	11	
STATE NUMBER: 23			
23. 1 ..SUBR	DEL, PUSH, 2, SUBR: 24	-1	
23. 2 %O		11	
STATE NUMBER: 24			
24. 1 ..SUBR	DEL, P, 1, : -1, P, 1, : -1, P, 1, : -1, P, 1, : -1, GD	-1	
24. 2 )	DEL, P, 1, : -1, P, 1, : -2, P, 2, SUBR: 26	24	
24. 3 %O	DEL, P, 1, : -1, P, 1, : -3, P, 2, SUBR: 26	11	
24. 4 %5	DEL, P, 1, : -1, P, 1, : -3, P, 2, SUBR: 26	11	
24. 5 %3	DEL, P, 2, SUBR: 25	11	
STATE NUMBER: 25			
25. 1 ..SUBR	DEL, P, 1, : -1, P, 1, : -1, P, 1, : -1, GD	-1	
25. 2 )	DEL, P, 1, : -2, PUSH, 2, SUBR: 26	25	
25. 3 %O	DEL, P, 1, : -3, PUSH, 2, SUBR: 26	11	
25. 4 %5		11	
STATE NUMBER: 26			
26. 1 ..SUBR	DEL, P, 1, : -1, GENNODE	-1	
26. 2 )	DEL, PUSH, 2, SUBR: 27	26	
26. 3 %3		11	
STATE NUMBER: 27			
27. 1 ..SUBR	DEL, GENNODE	-1	
27. 2 )		27	
STATE NUMBER: 28			
28. 1 ), (	DEL, POP, 2	28	
28. 2		12	
STATE NUMBER: 29			

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 30			
30. 1 (	PUSH, 2, 1e DEL	31	
STATE NUMBER: 31			
31. 1 ^	PUSH, 2, 1e: 1 DEL	36	
31. 2 (	PUSH, 2, 1e DEL	32	
31. 3	PUSH, 2, SUBR: 33	11	
STATE NUMBER: 32			
32. 1 (	PUSH, 2, 1e DEL	32	
32. 2	PUSH, 2, SUBR: 33	11	
STATE NUMBER: 33			
33. 1 ^	PUSH, 2, 1e: 1 DEL	37	
33. 2 )..(	POP, 2 DEL	33	
33. 3 =	PUSH, 2, 1e: 2 DEL	38	
33. 4 eREL	PUSH, 2, 1e: 2 DEL	34	
STATE NUMBER: 34			
34. 1 (	PUSH, 2, 1e DEL	34	
34. 2	PUSH, 2, SUBR: 35	11	
STATE NUMBER: 35			
35. 1 )..(	POP, 2 DEL	35	
35. 2 ..eREL	GENNODE	35	
35. 3 ^	GENNODE	35	
35. 4 ..eCMP	GENNODE	35	
35. 5 eCMP	PUSH, 2, 1e: 2 DEL	31	
35. 6 ..SUBR	-1	-1	
STATE NUMBER: 36			
36. 1 ^	PUSH, 2, 1e: 1 DEL	36	
36. 2 (	PUSH, 2, 1e DEL	32	
STATE NUMBER: 37			
37. 1 ^	PUSH, 2, 1e: 1 DEL	37	
37. 2 eREL	PUSH, 2, 1e: 2 DEL	34	
STATE NUMBER: 38			
38. 1 (	PUSH, 2, 1e DEL	39	
38. 2		34	
STATE NUMBER: 39			
39. 1 :A :S	PUSH, 1, 1e DEL	40	
39. 2		34	
STATE NUMBER: 40			
40. 1 )..(	DEL POP, 2 GENNODE	35	
40. 2 %O	DEL MULX, 1	41	

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 41 41. 1 :A::S	PUSH.1.1@DEL	42
STATE NUMBER: 42 42. 1 :). 42. 2 %O	DEL POP.2 ADDON.1@POP.1 GENNODE DEL ADDON.1@POP.1	35 41
STATE NUMBER: 43		
STATE NUMBER: 47 47. 1 (	DEL	48
STATE NUMBER: 48 48. 1 (	PUSH.2.1@DEL	54
STATE NUMBER: 49 49. 1 :@VIRT 49. 2 )	VIRTX.1.1@DEL GENENT DEL	52 50
STATE NUMBER: 50 50. 1 WHERE 50. 2	DEL PUSH.2.SUBR:51	30 52
STATE NUMBER: 51 51. 1	ATTWHR	52
STATE NUMBER: 52 52. 1 : 52. 2 BY 52. 3 ) 52. 4 %O	DEL DEL VIRTA PUSH.2.SUBR:52	53 53 53 48
STATE NUMBER: 53 53. 1 :.SUBR		-1
STATE NUMBER: 54 54. 1 :R 54. 2 @VIRT 54. 3 (	PUSH.1.1@DEL PUSH.1.1@DEL	49 49 100
STATE NUMBER: 55 55. 1 (	PUSH.2.1@DEL	56
STATE NUMBER: 56 56. 1 :R 56. 2 @VIRT 56. 3 (	PUSH.1.1@DEL PUSH.1.1@DEL	57 57 112

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 57	VIRTX,1,1e DEL	60
57.1 } .eVIRT	GENENT DEL	58
57.2 }		
STATE NUMBER: 58	DEL PUSH,2,SUBR:59	30
58.1 WHERE		60
58.2		
STATE NUMBER: 59	ATTWHR	60
59.1		
STATE NUMBER: 60		70
60.1 eSETOP		-1
60.2 ..SUBR	GENENT DEL	58
60.3 }		
STATE NUMBER: 61	PUSH,2,1e DEL	62
61.1 {		
STATE NUMBER: 62	PUSH,1,ce DEL	63
62.1 :R	PUSH,1,1e DEL	63
62.2 eVIRT		124
62.3 {		
STATE NUMBER: 63	VIRTX,1,1e DEL	65
63.1 } .eVIRT	GENENT DEL	64
63.2		
STATE NUMBER: 64	DEL PUSH,2,SUBR:65	30
64.1 WHERE		66
64.2		
STATE NUMBER: 65	ATTWHR	66
65.1		
STATE NUMBER: 66		-1
66.1 ..SUBR	GENENT DEL	64
66.2 }		
STATE NUMBER: 67		

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 70			
70. 1 CS	PUSH, 2, 1e DEL		71
70. 2			80
STATE NUMBER: 71			
71. 1 (	PUSH, 2, 1e DEL		72
STATE NUMBER: 72			
72. 1 -	DEL		73
STATE NUMBER: 73			
73. 1 :R	PUSH, 1, ce:V:C DEL		74
STATE NUMBER: 74			
74. 1 (	PUSH, 2, IND( DEL INDX, 1		75
74. 2			78
STATE NUMBER: 75			
75. 1 :R	PUSH, 1, 1e DEL		76
STATE NUMBER: 76			
76. 1 (	PUSH, 2, IND( DEL ADDON, 1e POP, 1		75
76. 2 )..IND(	ADDON, 1e POP, 1 DEL POP, 2		77
STATE NUMBER: 77			
77. 1 )..IND(	DEL POP, 2		77
77. 2			78
STATE NUMBER: 78			
78. 1 %O	PUSH, 2, 1e DEL		90
STATE NUMBER: 79			
STATE NUMBER: 80			
80. 1	PUSH, 2, 1e DEL		81
STATE NUMBER: 81			
81. 1 (	PUSH, 2, 1e DEL		82
STATE NUMBER: 82			
82. 1 :R	PUSH, 1, ce:V DEL		83
STATE NUMBER: 83			
83. 1 )	PUSH, 2, 1e DEL		61
83. 2 %O	PUSH, 2, 1e DEL		82
83. 3 (	PUSH, 2, IND( DEL INDX, 1		84
STATE NUMBER: 84			
84. 1 :R	PUSH, 1, 1e DEL		85



FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 85		
85. 1 (		86
85. 2 ) .. IND(		86
STATE NUMBER: 86		
86. 1 ) .. IND(	PUSH, 2, IND( ; DEL ; ADDON, 1 ; POP, 1	
86. 2	ADDON, 1 ; POP, 1 ; DEL ; POP, 2	
	DEL ; POP, 2	86
		87
STATE NUMBER: 87		
87. 1 )		61
87. 2 %O	PUSH, 2, 1 ; DEL	82
	PUSH, 2, 1 ; DEL	
STATE NUMBER: 88		
STATE NUMBER: 90		
90. 1 :R	PUSH, 1, C ; V ; DEL	91
STATE NUMBER: 91		
91. 1 (		92
91. 2	PUSH, 2, IND( ; DEL ; IND, 1	94
STATE NUMBER: 92		
92. 1 :R	PUSH, 1, C ; V ; DEL	93
STATE NUMBER: 93		
93. 1 (		92
93. 2 ) .. IND(	PUSH, 2, IND( ; DEL ; ADDON, 1 ; POP, 1	94
	ADDON, 1 ; POP, 1 ; DEL ; POP, 2	
STATE NUMBER: 94		
94. 1 ) .. IND(	DEL ; POP, 2	94
94. 2 )	PUSH, 2, 1 ; DEL	61
STATE NUMBER: 95		
STATE NUMBER: 100		
100. 1 .. (		101
100. 2	PUSH, 2, SUBR : 52	111
STATE NUMBER: 101		
101. 1 .. (		102
101. 2	PUSH, 2, SUBR : 52	110
STATE NUMBER: 102		
102. 1 .. (		103
102. 2	PUSH, 2, SUBR : 52	109
STATE NUMBER: 103		
103. 1 .. (		104
103. 2	PUSH, 2, SUBR : 52	108

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 104			
104. 1 ..{	POP.2		105
104. 2	PUSH.2,SUBR:52		107
STATE NUMBER: 105			
105. 1	PUSH.2,SUBR:52		106
STATE NUMBER: 106			
106. 1	PUSH.2,{		107
STATE NUMBER: 107			
107. 1	PUSH.2,{		108
STATE NUMBER: 108			
108. 1	PUSH.2,{		109
STATE NUMBER: 109			
109. 1	PUSH.2,{		110
STATE NUMBER: 110			
110. 1	PUSH.2,{		111
STATE NUMBER: 111			
111. 1			55
STATE NUMBER: 112			
112. 1 ..{	POP.2		113
112. 2	PUSH.2,SUBR:60		123
STATE NUMBER: 113			
113. 1 ..{	POP.2		114
113. 2	PUSH.2,SUBR:60		122
STATE NUMBER: 114			
114. 1 ..{	POP.2		115
114. 2	PUSH.2,SUBR:60		121

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 115			
115. 1 ..{	POP, 2		116
115. 2	PUSH, 2, SUBR: 60		120
STATE NUMBER: 116			
116. 1 ..{	POP, 2		117
116. 2	PUSH, 2, SUBR: 60		119
STATE NUMBER: 117			
117. 1	PUSH, 2, SUBR: 60		118
STATE NUMBER: 118			
118. 1	PUSH, 2, {		119
STATE NUMBER: 119			
119. 1	PUSH, 2, {		120
STATE NUMBER: 120			
120. 1	PUSH, 2, {		121
STATE NUMBER: 121			
121. 1	PUSH, 2, {		122
STATE NUMBER: 122			
122. 1	PUSH, 2, {		123
STATE NUMBER: 123			
123. 1			55
STATE NUMBER: 124			
124. 1 ..{	POP, 2		125
124. 2	PUSH, 2, SUBR: 66		135
STATE NUMBER: 125			
125. 1 ..{	POP, 2		126
125. 2	PUSH, 2, SUBR: 66		134
STATE NUMBER: 126			
126. 1 ..{	POP, 2		127
126. 2	PUSH, 2, SUBR: 66		133
STATE NUMBER: 127			
127. 1 ..{	POP, 2		128
127. 2	PUSH, 2, SUBR: 66		132
STATE NUMBER: 128			
128. 1 ..{	POP, 2		129
128. 2	PUSH, 2, SUBR: 66		131

FINITE-STATE-MACHINE RULES  
MATCH - ACTION - NEXT\_STATE

STATE NUMBER: 129 129. 1	PUSH, 2, SUBR: 66	130
STATE NUMBER: 130 130. 1	PUSH, 2, (	131
STATE NUMBER: 131 131. 1	PUSH, 2, (	132
STATE NUMBER: 132 132. 1	PUSH, 2, (	133
STATE NUMBER: 133 133. 1	PUSH, 2, (	134
STATE NUMBER: 134 134. 1	PUSH, 2, (	135
STATE NUMBER: 135 135. 1		55
STATE NUMBER: 136		
STATE NUMBER: 140 140. 1 (	DEL	141
STATE NUMBER: 141 141. 1 (	DEL	142
STATE NUMBER: 142 142. 1 @VIRT	PUSH, 1, 1@DEL	143
STATE NUMBER: 143 143. 1 )	DEL BY ENT POP, 1	144
STATE NUMBER: 144 144. 1 ) 144. 2	DEL PUSH, 2, SUBR: 145	53 11
STATE NUMBER: 145 145. 1	ATTBY POP, 1	146
STATE NUMBER: 146 146. 1 ) 146. 2 %O	DEL DEL PUSH, 2, SUBR: 145	53 11
STATE NUMBER: 147		

MACHINE DEFINITION COMPLETE

(A SAMPLE SESSION OF A VERY BASIC EXAMPLE)

IN THE FOLLOWING PAGES, A SAMPLE TERMINAL SESSION IS ILLUSTRATED. LINES WHICH BEGIN WITH '\$\$\$' ARE USER INPUT LINES, AND LINES WITH TEXT ENCLOSED IN A SET OF PARENTHESES ARE NEITHER USER INPUT NOR PROGRAM OUTPUT, THEY ARE DESCRIPTIONS OF THE ILLUSTRATION. ALL OTHER LINES ARE PROGRAM OUTPUT; HOWEVER, THE PRINTING OF THE TOKEN CHAIN, THE MACHINE ACTIVITIES, THE EXECUTION TREE, AND THE ENTITY SET TABLE MAY BE SUPPRESSED IN ACTUAL USE.

STATUS: CMS FILE "FILE ZZZ" CONTAINS THE FOLLOWING FOUR LINES:

```
DEFINE HEAVY AS WEIGHT > 300 ;  
ADHOC IO AS 1/GPA * 2500 ;  
RETRIEVE ( ( EMPLOYEE ) WHERE (HEAVY AND INDEX > 100 OR IO > 150) )  
BY ( ( VO ) NAME ) ;
```

(IN CMS)  
R:  
\$\$\$  
(SCREEN REFRESHED)

LOAD USINT (START MODUP

\*\*\*INFOPLEX DATA BASE MACHINE\*\*\*  
TYPE "VIR" FOR VIRTUAL INFORMATION PROCESSOR  
TYPE "REAL" FOR REAL INFORMATION PROCESSOR  
:  
\$\$\$  
(SCREEN REFRESHED) VIR

\*\*\*TRANSACTION BUFFER\*\*\*  
\*\*\*NO MORE\*\*\*

\*\*\*EXECUTION BUFFER\*\*\*  
\*\*\*NO MORE\*\*\*

FINPUT ZZZ

0 . . . . .  
. . . . .

\$\$\$  
(SCREEN REFRESHED)



READING TRANSACTION BUFFER FROM CMS FILE:  
"FILE 222"  
OLD TRANSACTION BUFFER CONTENT DELETED  
(SCREEN REFRESHED)

```

***TRANSACTION BUFFER***
0 DEFINE HEAVY AS WEIGHT > 300 ;
1 ADHOC IQ AS 1/GPA * 2500 ;
2 RETRIEVE ( { EMPLOYEE } WHERE (HEAVY AND INDEX > 100 OR IQ > 150) )
3 BY ( { VO } NAME ) ;
4 ***NO MORE***
..
***EXECUTION BUFFER***
***NO MORE***

```

166

RUNTRANS

\$\$\$  
(SCREEN REFRESHED)

HEAVY DEFINITION: SAVED IN MAIN  
TYPE "ENTER" KEY TO CONTINUE  
...  
(SCREEN REFRESHED)

"ENTER" KEY

0100 DEFINITION: SAVED IN ADHOC  
TYPE "ENTER" KEY TO CONTINUE  
\$\$\$  
(SCREEN REFRESHED)

"ENTER" KEY

```

RETRIEVE ( ( EMPLOYEE ) WHERE (HEAVY AND INDEX > 100 OR IO > 150) )
      BY ( ( VO ) NAME ) :
( LIST OF INPUT TOKENS )
TEST_TOKEN = $RETRIEVE$
TEST_CLASS = $:B$$
TEST_TOKEN = $($$
TEST_CLASS = $:D$$
TEST_TOKEN = $:D$$
TEST_CLASS = $:D$$
TEST_TOKEN = $EMPLOYEE$
TEST_CLASS = $:R$$
TEST_TOKEN = $:R$$
TEST_CLASS = $:D$$
TEST_TOKEN = $WHERE$
TEST_CLASS = $:R$$
TEST_TOKEN = $($$
TEST_CLASS = $:D$$
TEST_TOKEN = $WEIGHT$
TEST_CLASS = $:R$$
TEST_TOKEN = $>$
TEST_CLASS = $:B$$
TEST_TOKEN = $ 300$
TEST_CLASS = $:A$$
TEST_TOKEN = $AND$
TEST_CLASS = $:B$$
TEST_TOKEN = $INDEX$
TEST_CLASS = $:R$$
TEST_TOKEN = $>$
TEST_CLASS = $:B$$
TEST_TOKEN = $ 100$
TEST_CLASS = $:A$$
TEST_TOKEN = $OR$
TEST_CLASS = $:B$$
TEST_TOKEN = $
TEST_CLASS = $:A$$
TEST_TOKEN = $/$
TEST_CLASS = $:B$$
TEST_TOKEN = $GPA$
TEST_CLASS = $:R$$
TEST_TOKEN = $+$$
TEST_CLASS = $:B$$
TEST_TOKEN = $ 2500$
TEST_CLASS = $:A$$
TEST_TOKEN = $>$
TEST_CLASS = $:B$$
TEST_TOKEN = $
TEST_CLASS = $:A$$
TEST_TOKEN = $)$
TEST_CLASS = $:D$$
TEST_TOKEN = $)$
TEST_CLASS = $:D$$

```

TEST\_TOKEN -\$BY\$\$  
TEST\_CLASS -\$:B\$\$  
TEST\_TOKEN -\$:\$\$  
TEST\_CLASS -\$:D\$\$  
TEST\_TOKEN -\$:\$\$  
TEST\_CLASS -\$:D\$\$  
TEST\_TOKEN -\$VO\$\$  
TEST\_CLASS -\$:R\$\$  
TEST\_TOKEN -\$:\$\$  
TEST\_CLASS -\$:D\$\$  
TEST\_TOKEN -\$NAME\$\$  
TEST\_CLASS -\$:R\$\$  
TEST\_TOKEN -\$:\$\$  
TEST\_CLASS -\$:D\$\$  
TEST\_TOKEN -\$:\$\$  
TEST\_CLASS -\$:D\$\$  
MACHINE DEFINITION COMPLETE  
(END OF TOKEN CHAIN)  
(FINITE STATE MACHINE DEFINED)

(TRACE OF FINITE STATE TRANSITIONS)

TRANSIT: STATE = 1  
 INPUT = \$RETRIEVE\$\$  
 CLASS = \$:B\$\$  
 STK#1 = \$080S\$\$  
 STK#2 = \$080S\$\$  
 MATCHING = \$RETRIEVE\$\$  
 FIND\_MATCH = \$RETRIEVE\$\$  
 FIND\_ITEM = \$RETRIEVE\$\$  
 FIND\_CLASS = \$:B\$\$  
 SEP\_ITEM = \$RETRIEVE\$\$  
 FIND\_MATCH = \$\$\$  
 FIND\_ITEM = \$080S\$\$  
 FIND\_CLASS = \$\$\$  
 FIND\_MATCH = \$\$\$  
 FIND\_ITEM = \$080S\$\$  
 FIND\_CLASS = \$\$\$  
 ACTING = \$PUSH,2,10\$\$  
 ACTING = \$DEL\$\$  
 ACTING = \$PUSH,2,SUBR:2\$\$

TRANSIT: STATE = 47  
 INPUT = \$(\$\$  
 CLASS = \$:D\$\$  
 STK#1 = \$080S\$\$  
 STK#2 = \$SUBR:2\$\$  
 MATCHING = \$(\$\$  
 FIND\_MATCH = \$(\$\$  
 FIND\_ITEM = \$(\$\$  
 FIND\_CLASS = \$:D\$\$  
 SEP\_ITEM = \$(\$\$  
 FIND\_MATCH = \$\$\$  
 FIND\_ITEM = \$080S\$\$  
 FIND\_CLASS = \$\$\$  
 FIND\_MATCH = \$\$\$  
 FIND\_ITEM = \$SUBR:2\$\$  
 FIND\_CLASS = \$\$\$  
 ACTING = \$DEL\$\$

TRANSIT: STATE = 48  
 INPUT = \$(\$\$  
 CLASS = \$:D\$\$  
 STK#1 = \$080S\$\$  
 STK#2 = \$SUBR:2\$\$  
 MATCHING = \$(\$\$  
 FIND\_MATCH = \$(\$\$  
 FIND\_ITEM = \$(\$\$  
 FIND\_CLASS = \$:D\$\$  
 SEP\_ITEM = \$(\$\$  
 FIND\_MATCH = \$\$\$  
 FIND\_ITEM = \$080S\$\$  
 FIND\_CLASS = \$\$\$  
 FIND\_MATCH = \$\$\$

FIND\_ITEM -\$\$SUBR:2\$\$  
 FIND\_CLASS -\$\$\$  
 ACTING -\$\$PUSH,2,10\$\$  
 ACTING -\$\$DEL\$\$

TRANSIT: STATE = 54  
 INPUT -\$\$EMPLOYEE\$\$  
 CLASS -\$:R\$\$  
 STK#1 -\$\$BOS\$\$  
 STK#2 -\$:I\$\$  
 MATCHING -\$:R\$\$  
 FIND\_MATCH -\$:R\$\$  
 FIND\_ITEM -\$\$EMPLOYEE\$\$  
 FIND\_CLASS -\$:R\$\$  
 FIND\_MATCH -\$\$\$  
 FIND\_ITEM -\$\$BOS\$\$  
 FIND\_CLASS -\$\$\$  
 FIND\_MATCH -\$\$\$  
 FIND\_ITEM -\$:I\$\$  
 FIND\_CLASS -\$\$\$  
 ACTING -\$\$PUSH,1,10\$\$  
 ACTING -\$\$DEL\$\$

TRANSIT: STATE = 49  
 INPUT -\$:I\$\$  
 CLASS -\$:D\$\$  
 STK#1 -\$\$EMPLOYEE\$\$  
 STK#2 -\$:I\$\$  
 MATCHING -\$:VIRT\$\$  
 FIND\_MATCH -\$:I\$\$  
 FIND\_ITEM -\$:I\$\$  
 FIND\_CLASS -\$:D\$\$  
 SEP\_ITEM -\$:I\$\$  
 FIND\_MATCH -\$\$VIRT\$\$  
 FIND\_ITEM -\$\$EMPLOYEE\$\$  
 FIND\_CLASS -\$\$\$  
 SEP\_ITEM -\$\$V0,V1,V2,V3,V4,V5,V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V1,V2,V3,V4,V5,V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V2,V3,V4,V5,V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V3,V4,V5,V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V4,V5,V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V5,V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V6,V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V7,V8,V9\$\$  
 SEP\_ITEM -\$\$V8,V9\$\$  
 SEP\_ITEM -\$\$V9\$\$  
 MATCHING -\$:I\$\$  
 FIND\_MATCH -\$:I\$\$  
 FIND\_ITEM -\$:I\$\$  
 FIND\_CLASS -\$:D\$\$  
 SEP\_ITEM -\$:I\$\$  
 FIND\_MATCH -\$\$\$  
 FIND\_ITEM -\$\$EMPLOYEE\$\$



FIND\_CLASS -\$\$\$  
FIND\_MATCH -\$\$\$  
FIND\_ITEM -\$(  
FIND\_CLASS -\$\$\$  
ACTING -\$GENENT\$  
ACTING -\$DEL\$

TRANSIT: STATE = 50  
( ..... )  
( ..... )  
( ..... )  
(MORE OF THE SAME)  
( ..... )  
( ..... )  
( ..... )

(FROM THIS POINT ON ONLY THE TRANSFER FROM STATE TO STATE WILL BE SHOWN)

TRANSIT: STATE = 30  
TRANSIT: STATE = 31  
TRANSIT: STATE = 11  
TRANSIT: STATE = 14  
TRANSIT: STATE = 12  
TRANSIT: STATE = 33  
TRANSIT: STATE = 34  
TRANSIT: STATE = 11  
TRANSIT: STATE = 12  
TRANSIT: STATE = 35  
TRANSIT: STATE = 35  
TRANSIT: STATE = 31  
TRANSIT: STATE = 11  
TRANSIT: STATE = 14  
TRANSIT: STATE = 12  
TRANSIT: STATE = 33  
TRANSIT: STATE = 34  
TRANSIT: STATE = 11  
TRANSIT: STATE = 12  
.....)  
(MORE OF THE SAME)  
(.....)  
(SKIP TO NEAR THE END)

TRANSIT: STATE = 53

INPUT = \$:\$\$

CLASS = \$:D\$\$

STK#1 = \$: 1\$\$

STK#2 = \$SUBR:2\$\$

MATCHING = \$:..SUBR\$\$

FIND\_MATCH = \$:\$\$

FIND\_ITEM = \$:\$\$

FIND\_CLASS = \$:D\$\$

FIND\_MATCH = \$:\$\$

FIND\_ITEM = \$: 1\$\$

FIND\_CLASS = \$:\$\$

FIND\_MATCH = \$SUBR\$\$

FIND\_ITEM = \$SUBR:2\$\$

FIND\_CLASS = \$:\$\$

SEP\_ITEM = \$SUBR\$\$

TRANSIT: STATE = 2

INPUT = \$:\$\$

CLASS = \$:D\$\$

STK#1 = \$: 1\$\$

STK#2 = \$RETRIEVE\$\$

MATCHING = \$:..RETRIEVE\$\$

FIND\_MATCH = \$:\$\$

FIND\_ITEM = \$:\$\$

FIND\_CLASS = \$:D\$\$

SEP\_ITEM = \$:\$\$

FIND\_MATCH = \$:\$\$

FIND\_ITEM = \$: 1\$\$

FIND\_CLASS = \$:\$\$

FIND\_MATCH = \$RETRIEVE\$\$

FIND\_ITEM = \$RETRIEVE\$\$

FIND\_CLASS = \$:\$\$

SEP\_ITEM = \$RETRIEVE\$\$

(END OF STATE TRANSITIONS)

(INPUT SUCCESSFULLY PARSED)

(EXECUTION TREE AND TABLE ON PAGES TO FOLLOW)

```

( EXECUTION TREE )

X TREE LOC = 1
LABEL = >
CHILD = 2
LINK = 4
X TREE LOC = 2
LABEL =
CHILD = 0
LINK = 3
X TREE LOC = 3
LABEL = 300:A
CHILD = 0
LINK = 0
X TREE LOC = 4
LABEL = 1:NTH
CHILD = 0
LINK = 0
X TREE LOC = 5
LABEL = >
CHILD = 2
LINK = 8
X TREE LOC = 6
LABEL =
CHILD = 0
LINK = 7
X TREE LOC = 7
LABEL = 100:A
CHILD = 0
LINK = 0
X TREE LOC = 8
LABEL = 2:NTH
CHILD = 0
LINK = 0
X TREE LOC = 9
LABEL = AND
CHILD = 2
LINK = 1
X TREE LOC = 10
LABEL =
CHILD = 0
LINK = 5
X TREE LOC = 11
LABEL = /
CHILD = 2
LINK = 14
X TREE LOC = 12
LABEL =
CHILD = 0
LINK = 13

```

XTREE\_LOC = 13  
 LABEL = 3:NTH  
 CHILD = 0  
 LINK = 0  
 XTREE\_LOC = 14  
 LABEL = 1:A  
 CHILD = 0  
 LINK = 0  
 XTREE\_LOC = 15  
 LABEL = \*  
 CHILD = 2  
 LINK = 11  
 XTREE\_LOC = 16  
 LABEL =  
 CHILD = 0  
 LINK = 17  
 XTREE\_LOC = 17  
 LABEL = 2500:A  
 CHILD = 0  
 LINK = 0  
 XTREE\_LOC = 18  
 LABEL = >  
 CHILD = 2  
 LINK = 15  
 XTREE\_LOC = 19  
 LABEL =  
 CHILD = 0  
 LINK = 20  
 XTREE\_LOC = 20  
 LABEL = 150:A  
 CHILD = 0  
 LINK = 0  
 XTREE\_LOC = 21  
 LABEL = OR  
 CHILD = 2  
 LINK = 9  
 XTREE\_LOC = 22  
 LABEL =  
 CHILD = 0  
 LINK = 18  
 (END OF EXECUTION TREE)  
 (ENTITY SET TABLE TO FOLLOW)

ENT\_NAME: EMPLOYEE  
 ENT\_DEPTH: 0  
 ATTRIBUTE: 1  
   SING\_OCC  
   A\_PARENT:  
   USES: WEIGHT  
 ATTRIBUTE: 2  
   SING\_OCC  
   A\_PARENT:  
   USES: INDEX  
 ATTRIBUTE: 3  
   SING\_OCC  
   A\_PARENT:  
   USES: GPA  
 ATTRIBUTE: 4  
   SING\_OCC  
   A\_PARENT:  
   USES: NAME  
 TYPE "ENTER" KEY TO CONTINUE  
 \$\$\$  
 (SCREEN REFRESHED)

"ENTER" KEY

```

***TRANSACTION BUFFER***
O DEFINE HEAVY AS WEIGHT > 300 ;
1 ADHOC IQ AS 1/GPA * .2500 ;
2 DEFINE INDEX AS SALARY + ID * DEPTNUM - WEIGHT / (SECNUM 1 2) ;
3 RETRIEVE ( { EMPLOYEE } WHERE (HEAVY AND INDEX > 100 OR IQ > 150) )
4 BY ( { VO } NAME) ;

```

\*\*\*EXECUTION BUFFER\*\*\*  
\*\*\*NO MORE\*\*\*

**TERMINATE**

(SCREEN REFRESHED)

R:  
(BACK TO CMS)



